READYNOW

AZUL'S JIT / AOT HYBRID

DOUGLAS Q. HAWKINS

LEAD JIT DEVELOPER

AZUL SYSTEMS

@dougqh

dhawkins@azul.com

ACKNOWLEDGEMENTS

Compiler Team

Runtime Team

GC Team

Perf Team

QA

Especially…

- Ivan Krylov
- Igor Laevsky
- Artur Pilipenko
- Roman Shirokiy
- Vladimir Voskeresensky

ABOUT AZUL ZING®

Known for C4 “Pauseless” GC

Derived from HotSpot

Popular with Financial Customers

READYNOW'S ORIGINAL PURPOSE

Originated from requests from our existing financial customers

Expanding out into other user applications

Eliminate Warm-up
First transaction is critical

No loss of peak performance

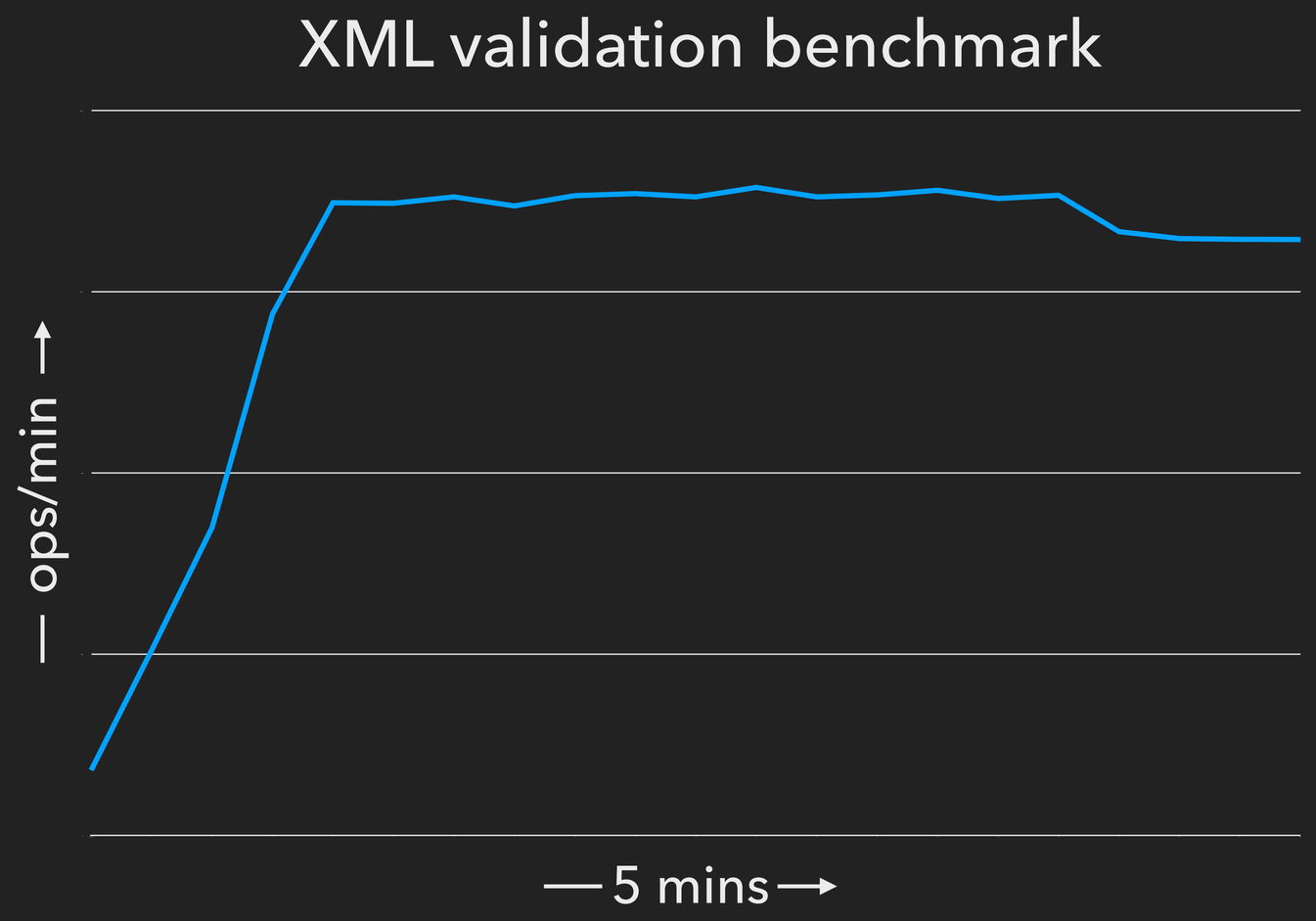
Eliminate volatility

No artificial load

START FAST!

STAY FAST!

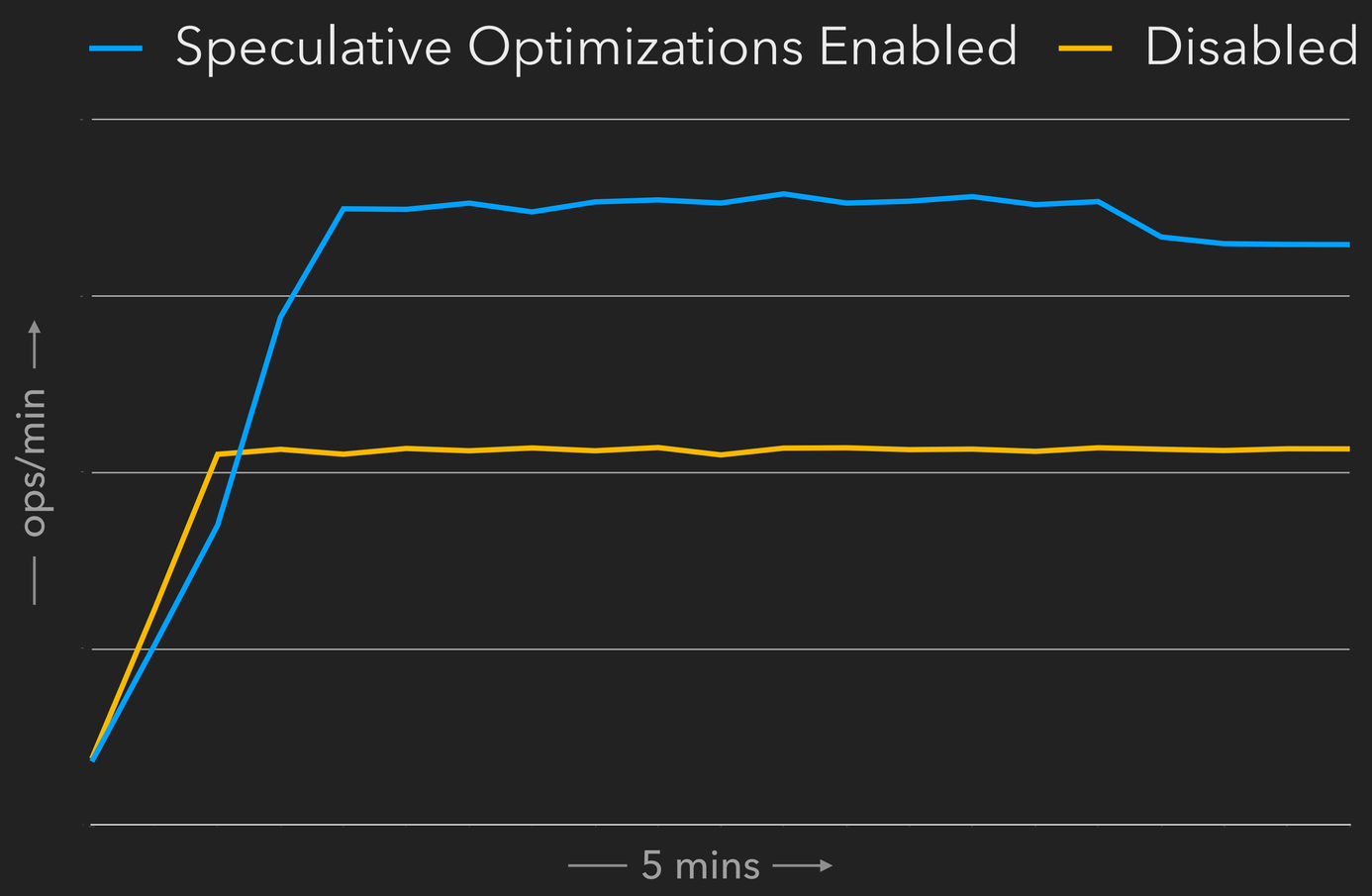
START WARMED-UP



Just-in-Time is
Slightly Too Late

Need Something…
Proactive
Not Reactive

~~NO~~ MINIMAL LOSS OF THROUGHPUT



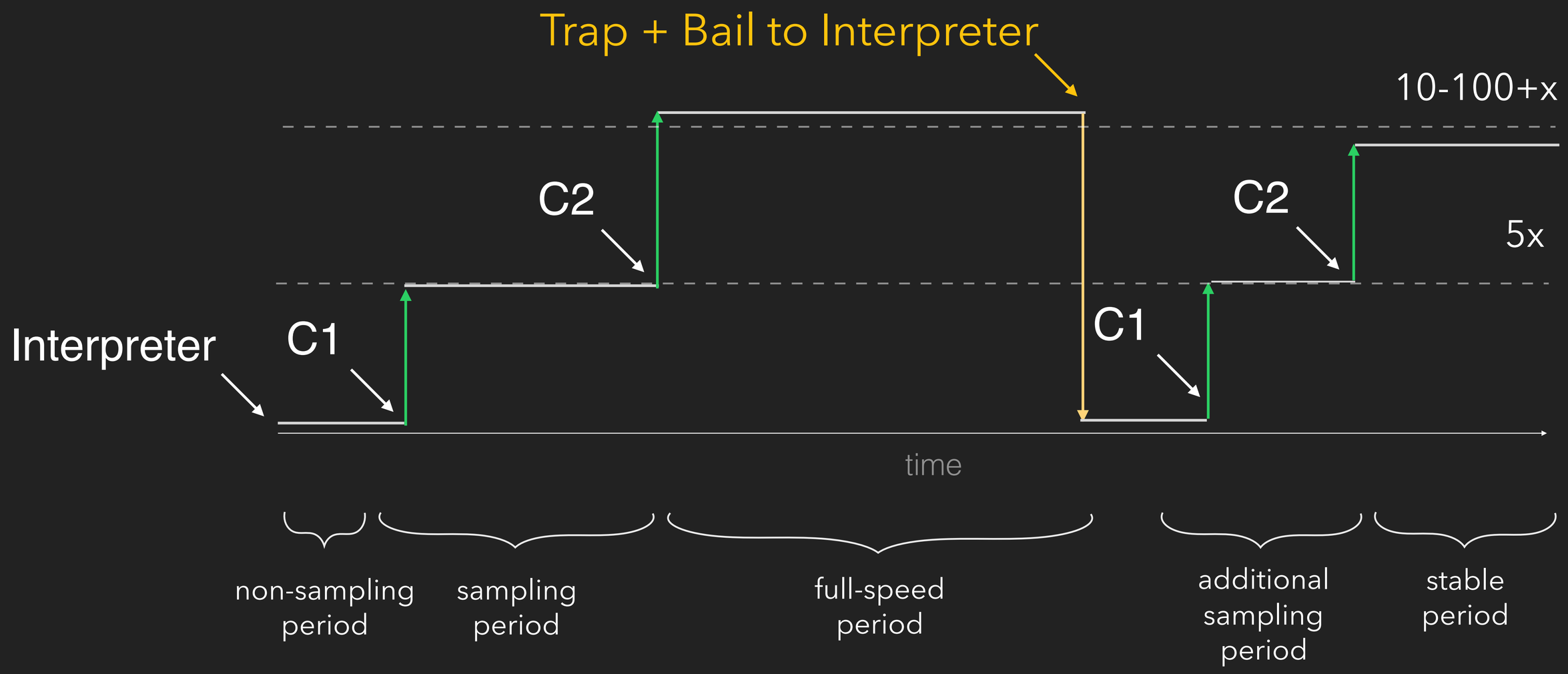
+30%

≅80*

Discarded Compilations

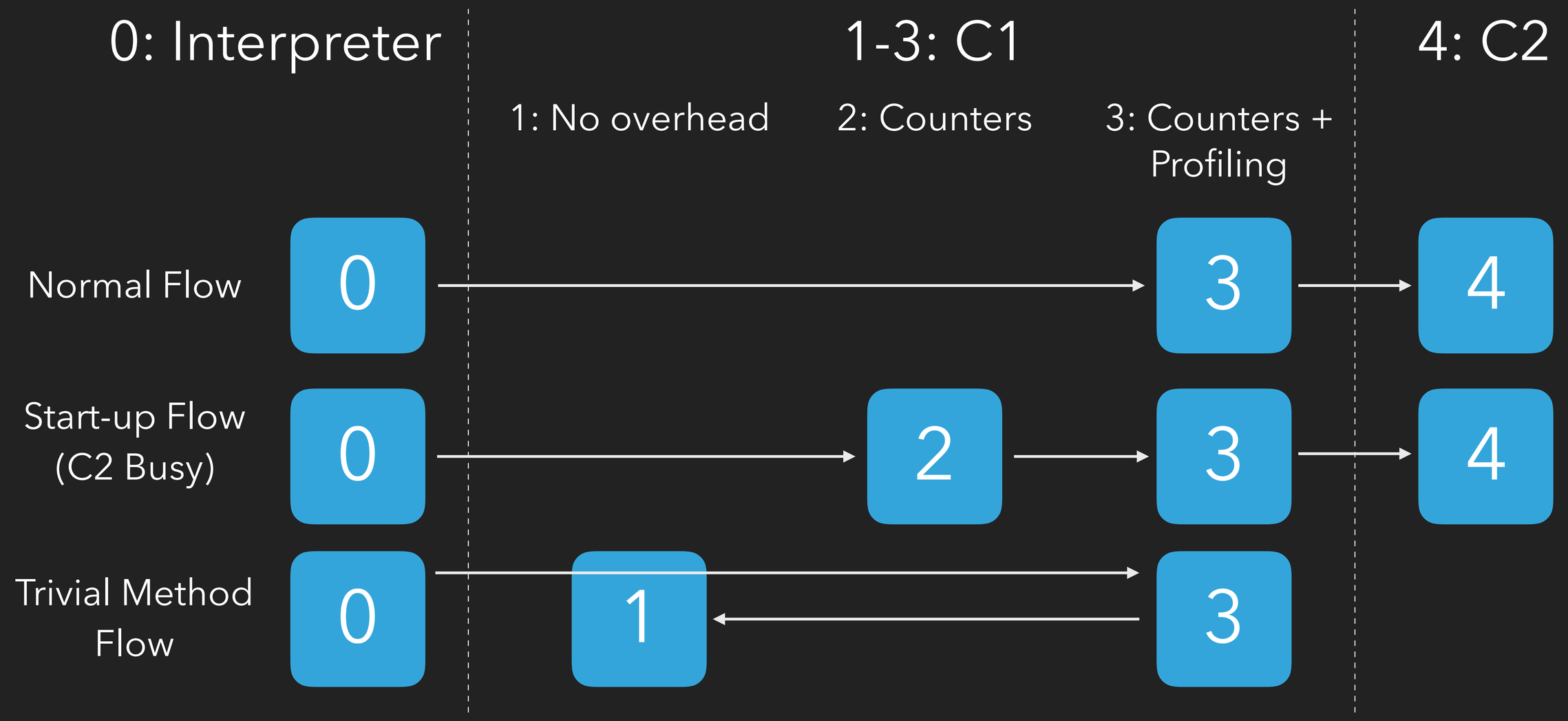
* out of 800 optimized tier compilations

SIMPLIFIED CYCLE OF OPENJDK



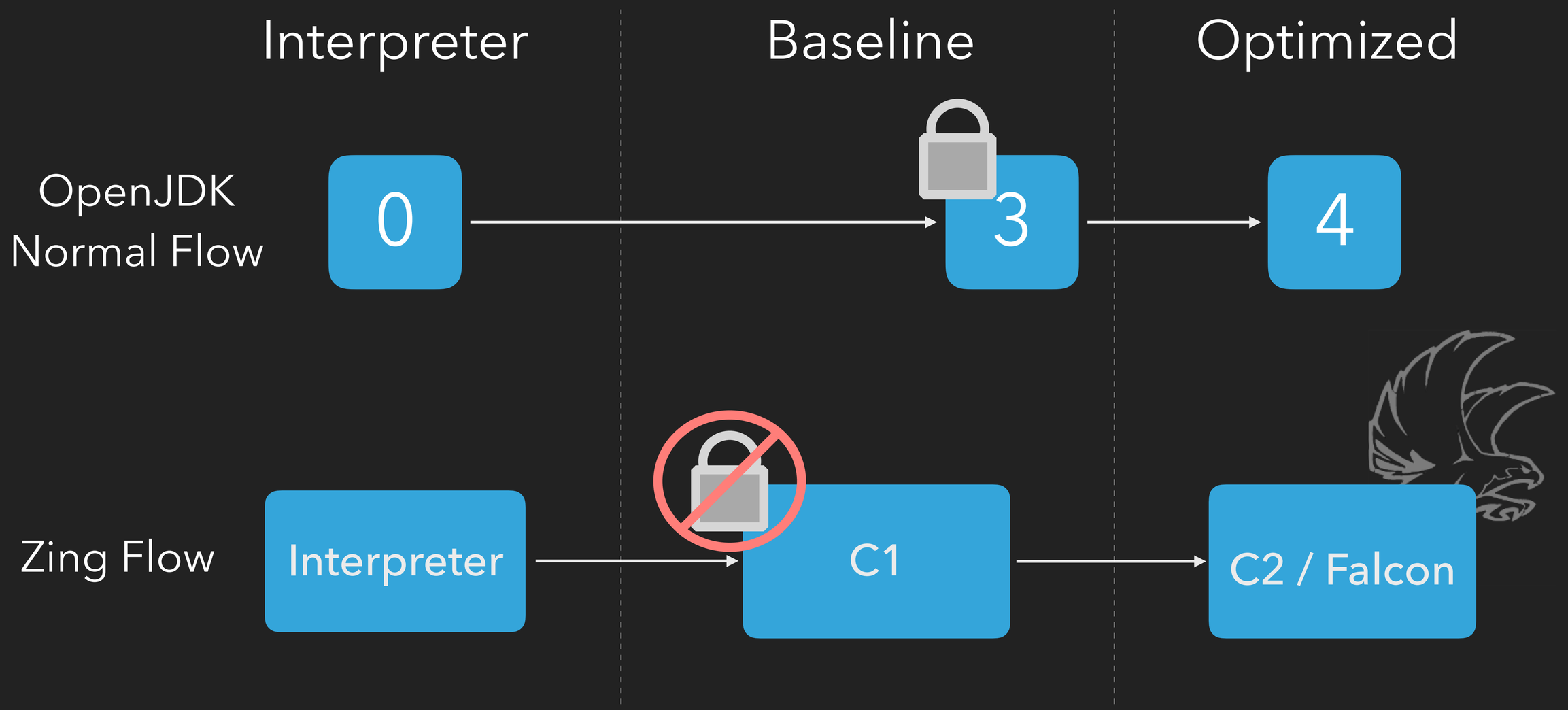
https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/

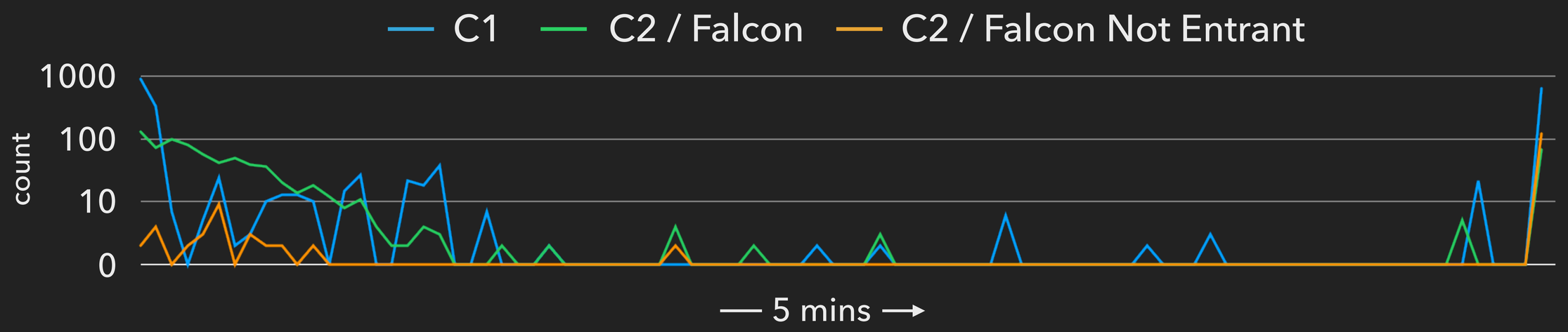
ACTUAL OPENJDK TIERING SYSTEM



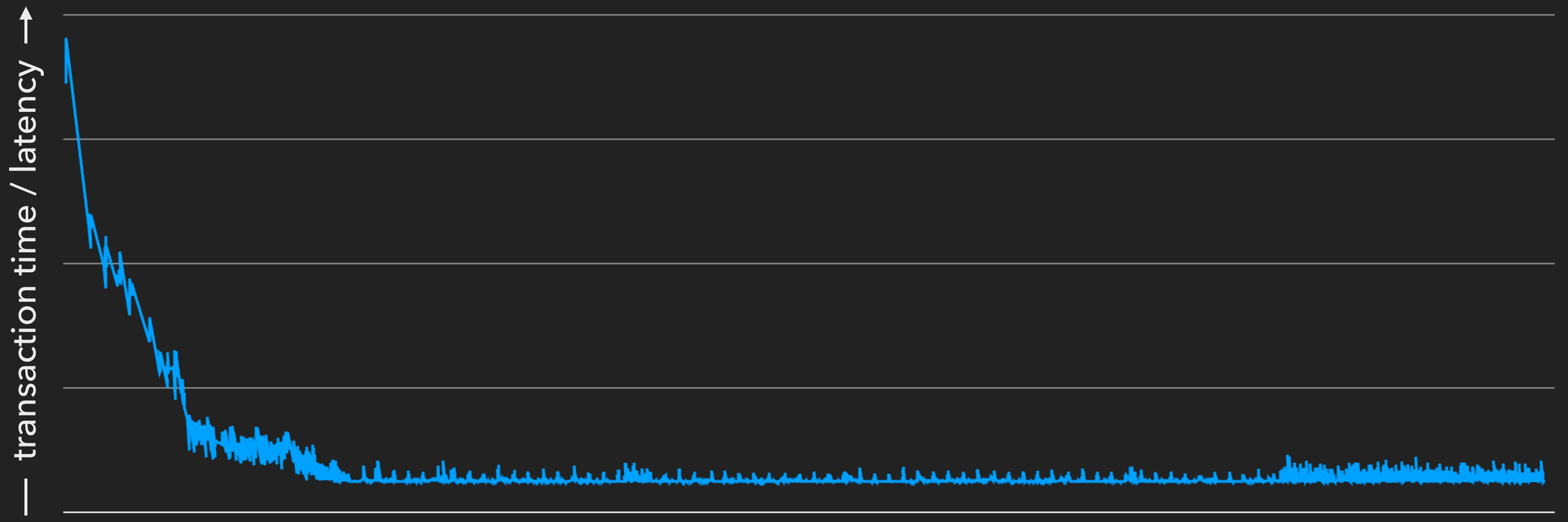
https://www.slideshare.net/maddocig/tiered

ZING TIERING SYSTEM

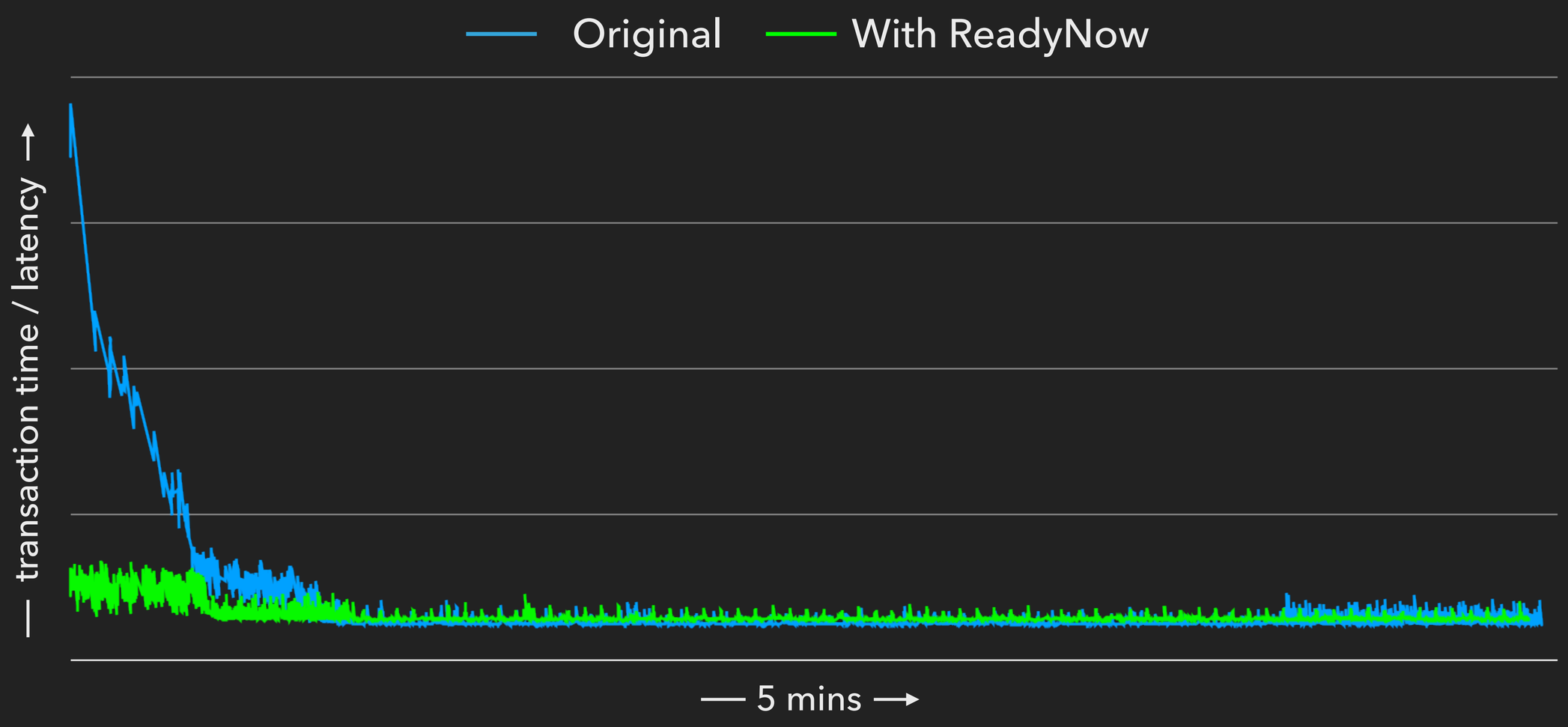




Invidual XML validation results



EVENTUALLY, WE'LL SEE...

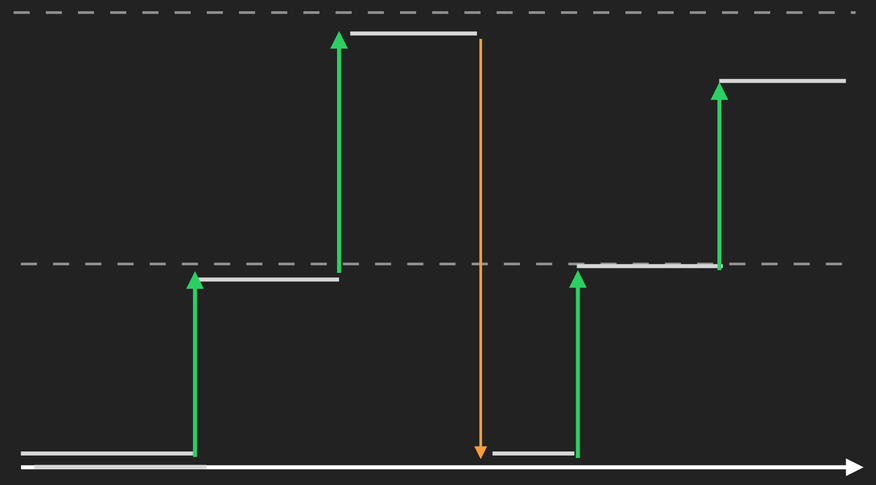


UNCONVENTIONAL APPROACH

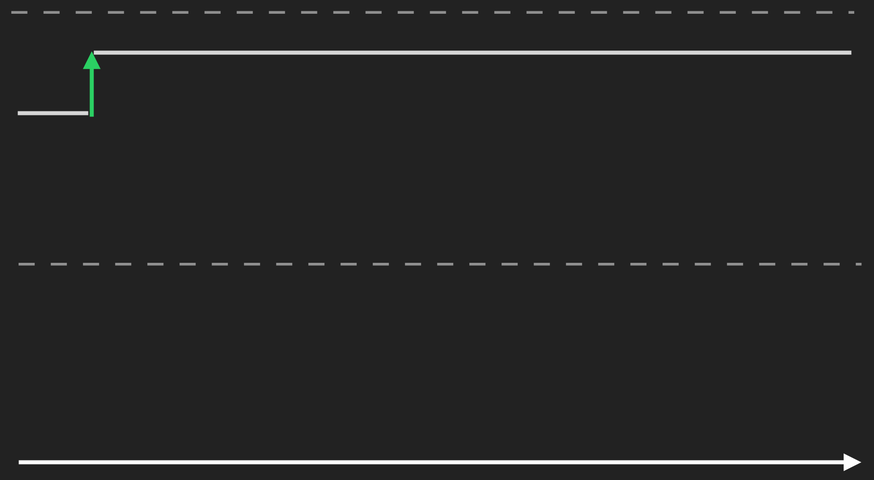
BUT "NATURAL" FOR HOTSPOT

AT A VERY *HIGH* LEVEL

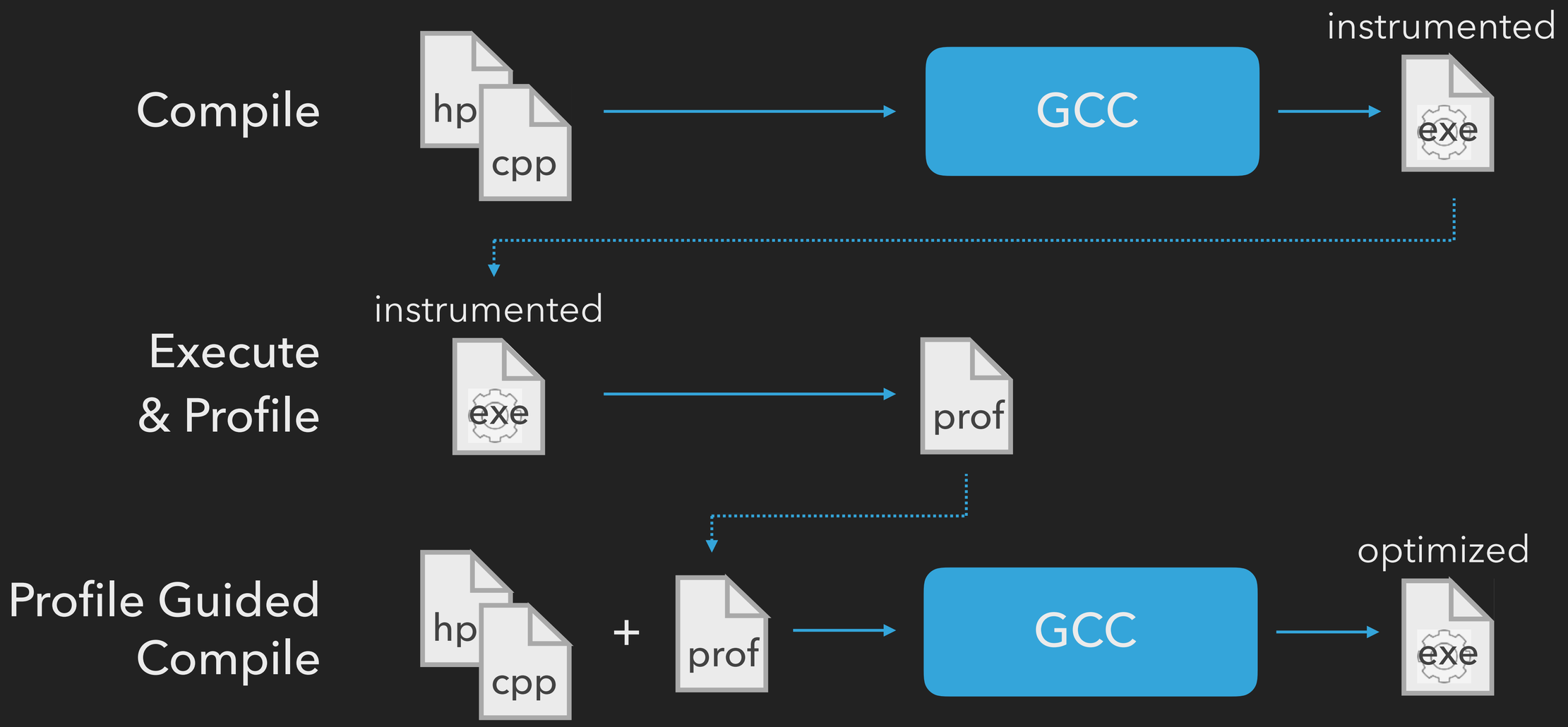
ORIGINAL RUN



SUBSEQUENT RUNS



MORE LIKE TRADITIONAL
PROFILE GUIDED OPTIMIZATION



READYNOW ISN'T...

AOT

JIT Cache

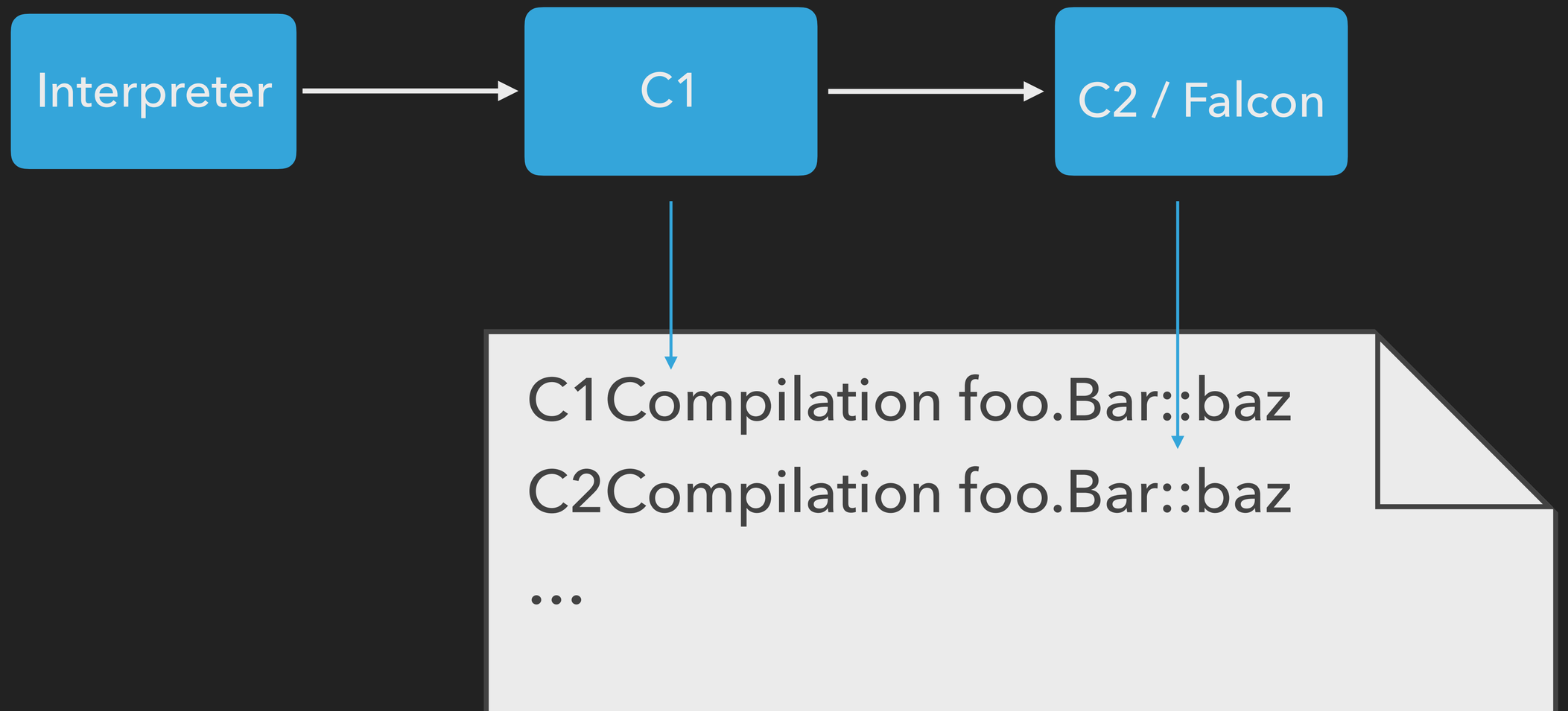
READYNOW IS...

Compilation Log

Profile Cache

Add-on to existing compilers: C1 & C2

COMPILATION LOG



CONCATENATIVE TRANSACTION LOG

Concatenative transaction log
UPSERT-s into a database

Grows with each compilation

Plain text - uncompressed

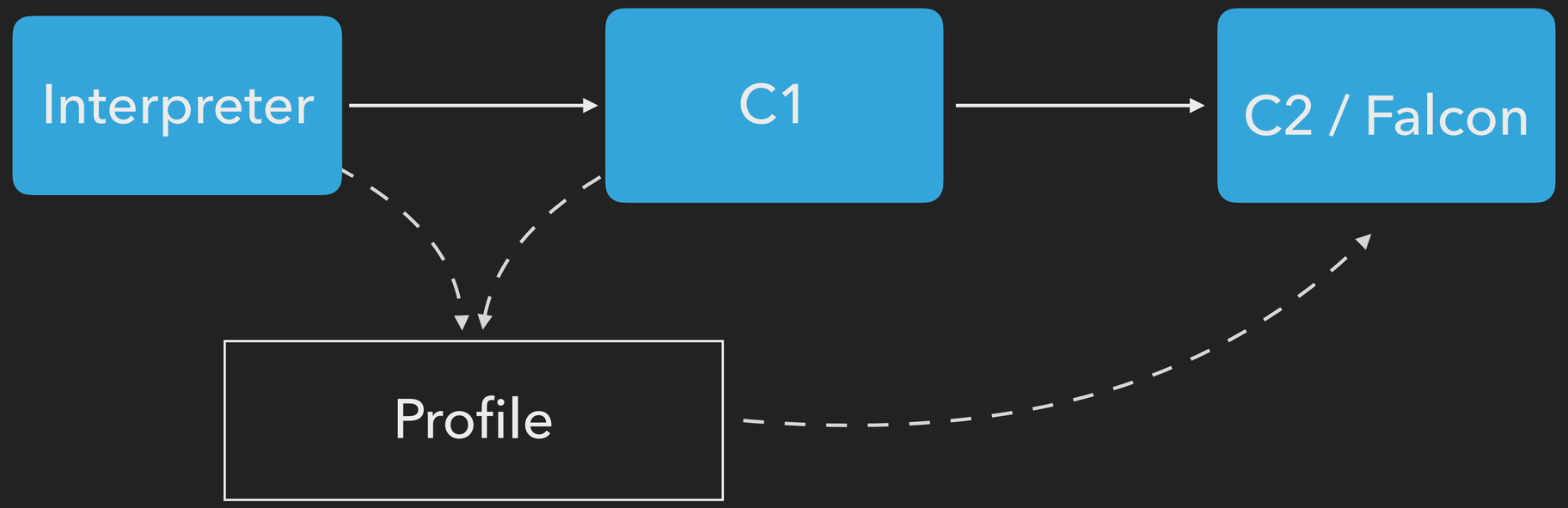
Backwards Compatible

Accidentally similar to CI Replay

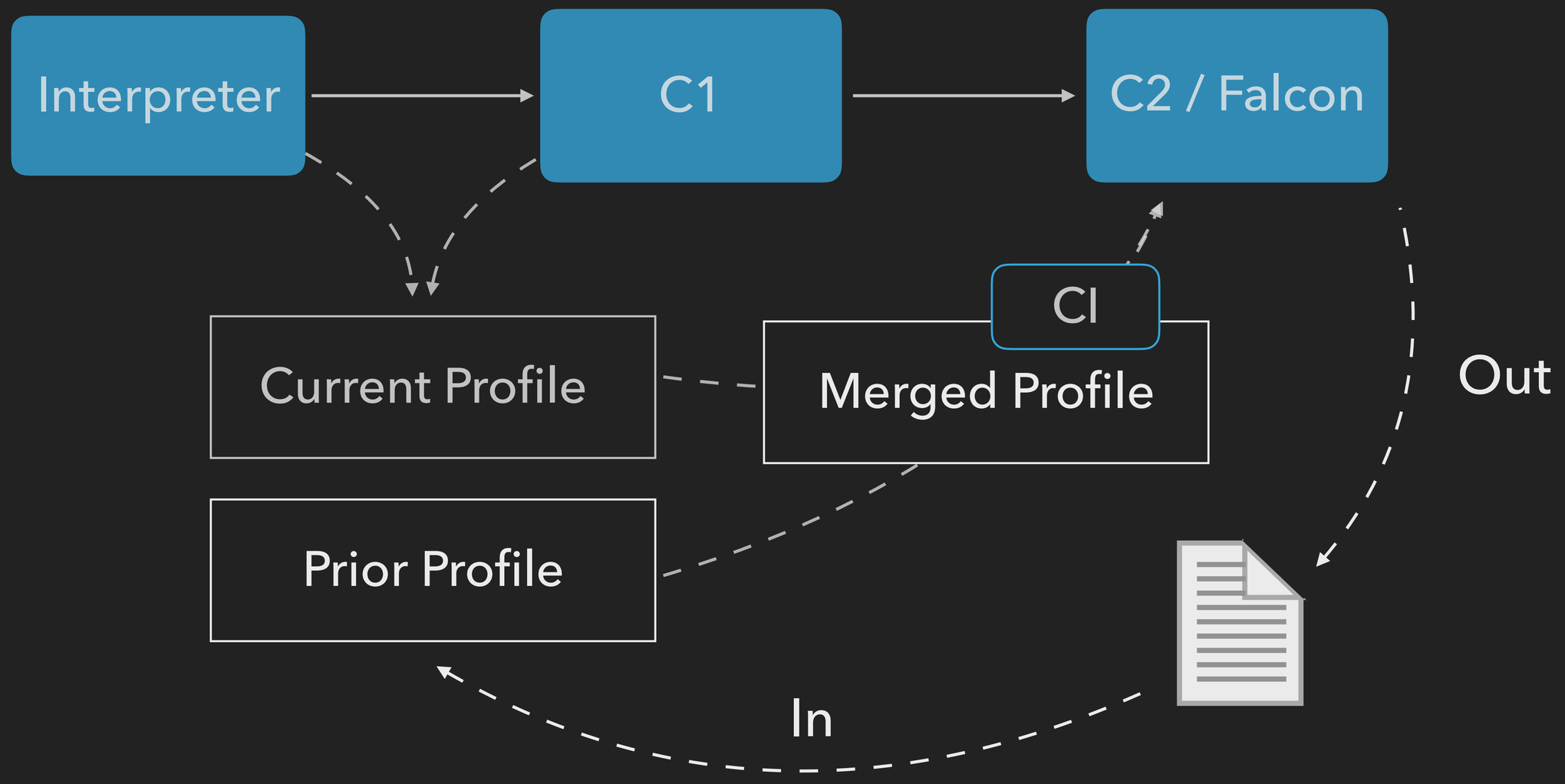
10-100 MiB on disk

5-30 MiB from native memory

ZING PROFILING



READYNOW PROFILE CACHE



SIMILAR PROFILE DATA TO OPENJDK

OPENJDK

LOG COMPILATION

READYNOW

PROFILE LOG

Branches

```
<branch
  target_bci='17'
  cnt='5800'
  taken='0'
  not_taken='5800'
  prob='never'/>
```

```
Branch compId methodId
  17
  5800
  0
  5800
```

Calls

```
<klass id='780' name=‘Square' …/>
<klass id='781' name=‘Sqrt’ …/>
<call method='783'
  count='23161'
  receiver='780' receiver_count=‘19901'
  receiver2='781' receiver2_count=‘3260'
  …/>
```

```
Class 780 Square
Class 781 Sqrt
Invoke compId methodId
  23161
  { { 780 19901 }
    { 781 3260 } }
```

https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/

MERGE PROFILES

Live
Call Histogram

Foo	7500
Bar	1500
Baz	1000

Recorded
Call Histogram

Quux	7500
Bar	1500
Foo	1000

Merged
Call Histogram

Foo	max(7500, 1000)
Bar	max(1500, 1500)
Baz	1000
Quux	7500

Prefer *Live* Data
over Recorded Data

Idempotent

AVOID COMPILE / DEOPT STORM

-XX:+PrintCompilation

```
 612   24 % !   3       …UninitializedForever::main @ 5 (25 bytes)
 621   25   !   3       …UninitializedForever::main (25 bytes)
 952   26 % !   4       …UninitializedForever::main @ 5 (25 bytes)
 953   24 % !   3       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
1024   26 % !   4       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
1033   27 % !   3       …UninitializedForever::main @ 5 (25 bytes)
1405   28 % !   4       …UninitializedForever::main @ 5 (25 bytes)
…
7339   55 % !   3       …UninitializedForever::main @ 5 (25 bytes)
7690   56 % !   4       …UninitializedForever::main @ 5 (25 bytes)
7690   55 % !   3       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
7761   56 % !   4       …UninitializedForever::main @ -2 (25 bytes)   made not entrant
7769   57 % !   3       …UninitializedForever::main @ 5 (25 bytes)
```


https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/ http://blog.joda.org/2011/08/printcompilation-jvm-flag.html

SIMPLE CONCEPT

NOT SO SIMPLE TO IMPLEMENT

NOT SO SIMPLE, THANKS TO

JAVA LANGUAGE

AND

JAVA ECOSYSTEM

FROM MY PERSPECTIVE

JAVA IS A DYNAMIC LANGUAGE

Dynamically Loaded

Lazy Initialized

Dynamically Dispatched

Runtime Checked

http://cr.openjdk.java.net/~twisti/slides/JVMLS%202015%20-%20Java%20Goes%20AOT.pdf

UNLOADED

JIT DOESN'T KNOW...

Fields

Methods

Parent Class

Interfaces

⋮

Anything

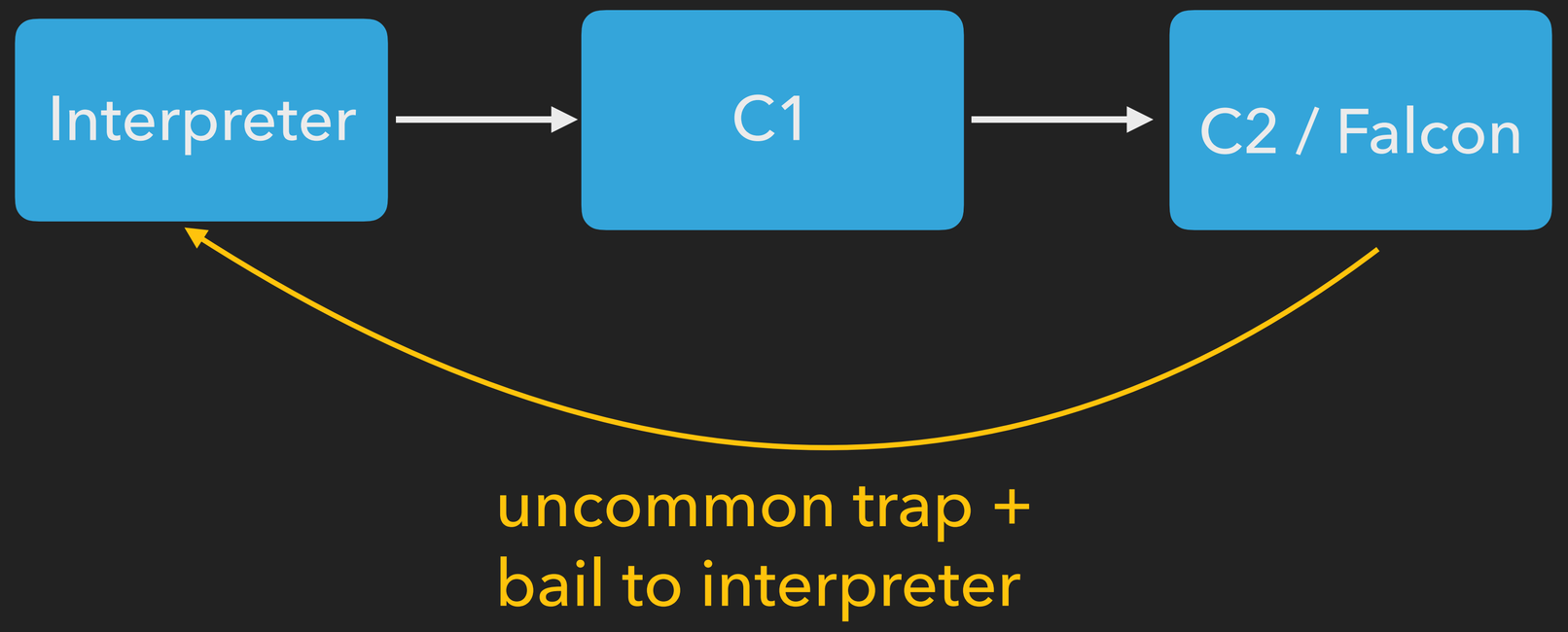
```
if ( always ) {
  return Class1.getStatic();
} else {
  return Class2.getStatic();
}
```

```
if ( always ) {
  return Class1.getStatic();
} else {
  uncommon_trap(:unloaded);
}
```

Give Up!

https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/

BAIL TO INTERPRETER

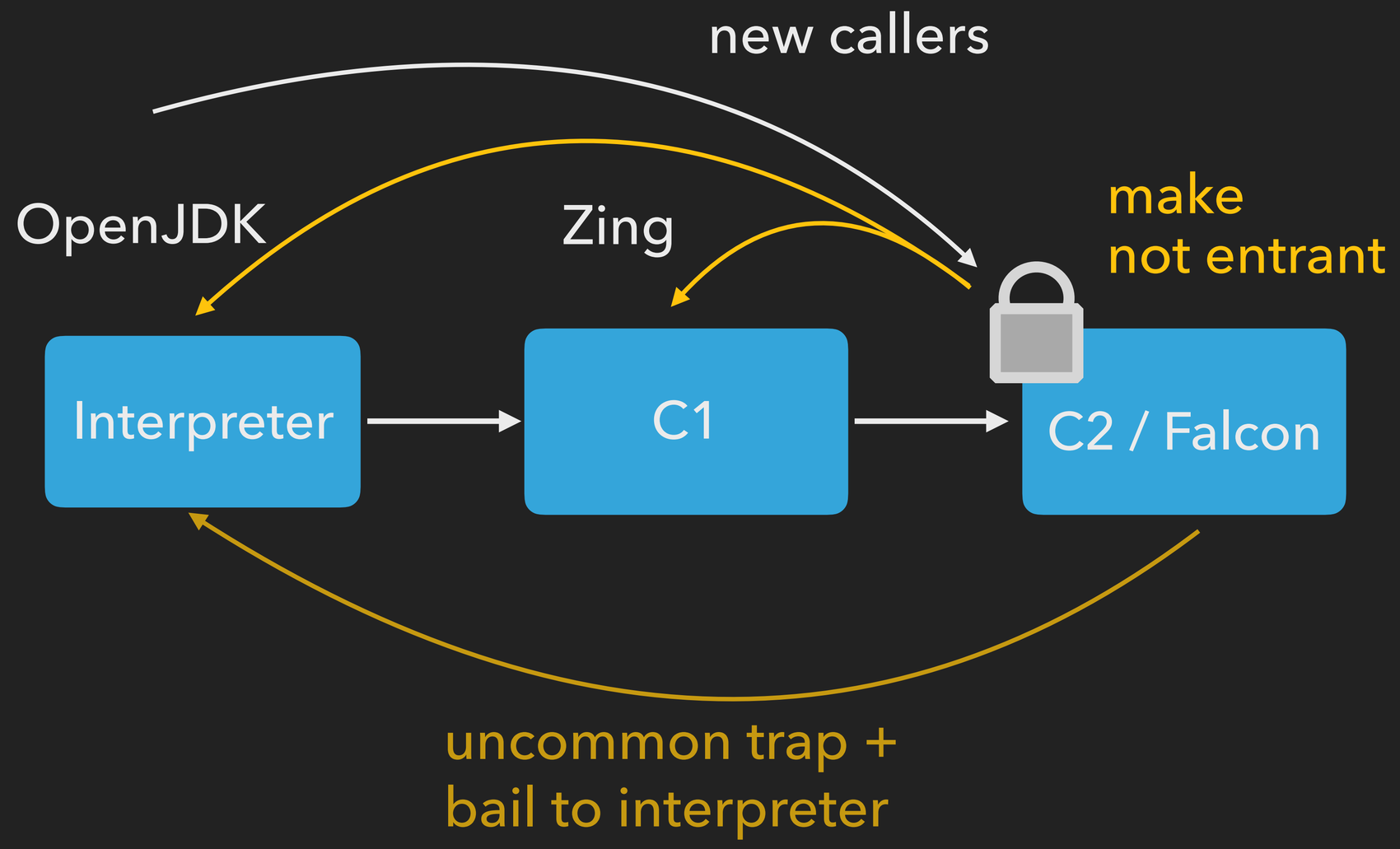


LOG COMPILATION

```
<uncommon_trap
  thread='7171'
  compile_id='24'
  compile_kind='osr'
  compiler='C2'
  level='4'
  reason='uninitialized'
  action='reinterpret'
  stamp='1.038'>
 …
</uncommon_trap>
```


https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/ https://wiki.openjdk.java.net/display/HotSpot/LogCompilation+overview

BAIL INTERPRETER + MAKE NOT ENTRANT



LOG COMPILATION

```
<make_not_entrant
  thread='7171'
  compile_id='24'
  compile_kind='osr'
  compiler='C2'
  level='4'
  stamp='1.038' />
```


https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/ https://wiki.openjdk.java.net/display/HotSpot/LogCompilation+overview

DEOPT STATISTICS

Traps	STW Deopts	Not Entrant	Recompiled
163	33	213	84

Making a Hot Method Not Entrant

10-100ms Latency Event

LAZY INITIALIZED

INITIALIZE CLASS AT FIRST...

Static Field Access

Static Method Call

New

Initialization of Child

http://cr.openjdk.java.net/~twisti/slides/JVMLS%202015%20-%20Java%20Goes%20AOT.pdf

INITIALIZATION CHECKING

```
MyClass.getStatic()
```

```
if ( !vm.is_init(MyClass)) {
  vm.init(MyClass);
}
MyClass.getStatic()
```

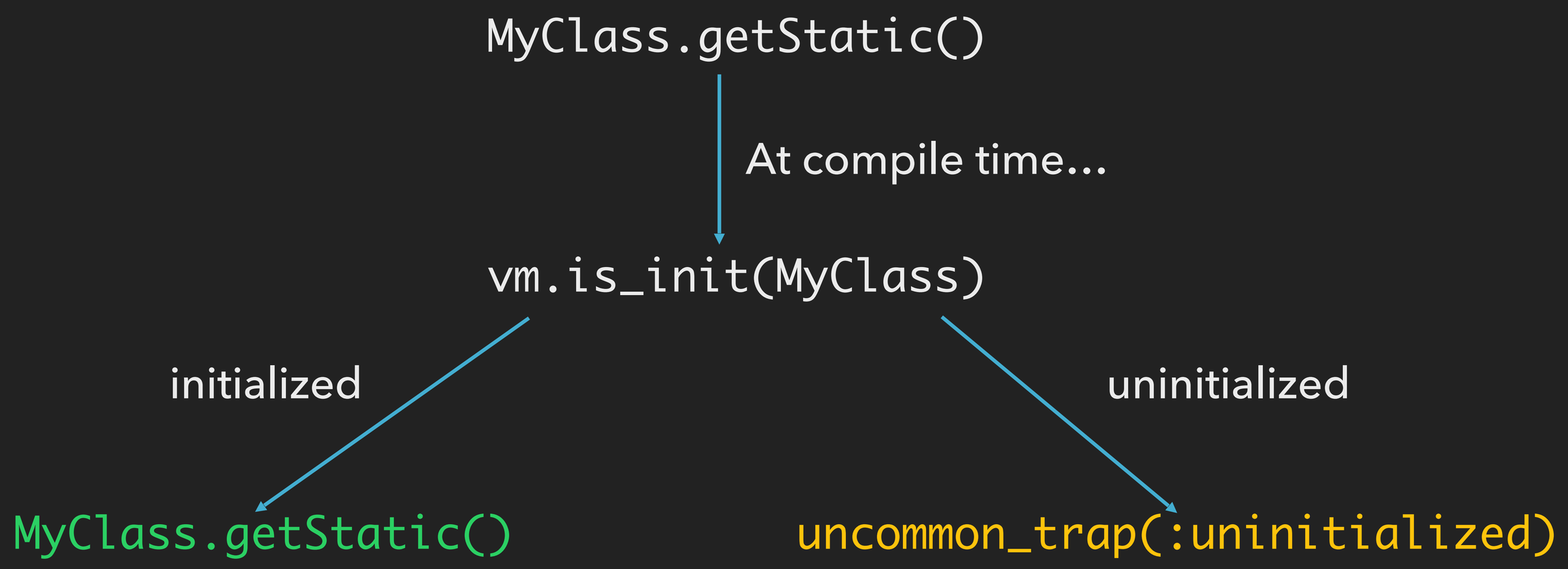
Bloats size by 20%

Slows down by 10+%

http://joeduffyblog.com/2015/12/19/safe-native-code/

JIT RUNS LATE

MOST CLASSES ARE ALREADY INITIALIZED



https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/

SPECULATION ALSO LEADS TO TRAPS

```
static final void hotMethod() {
  if ( thing == null )
    System.out.print("always");
  else
    System.out.print(“never");
}
```

```
static final void hotMethod() {
  if ( thing == null )
    System.out.print(“always");
  else
    uncommon_trap(:unstable_if);
}
```

trap + bail

```
Branch comp methodId
  17
  5800
  0
  5800
```

```
Branch comp methodId
  17
  5800
  1
  5800
```

update untaken branch

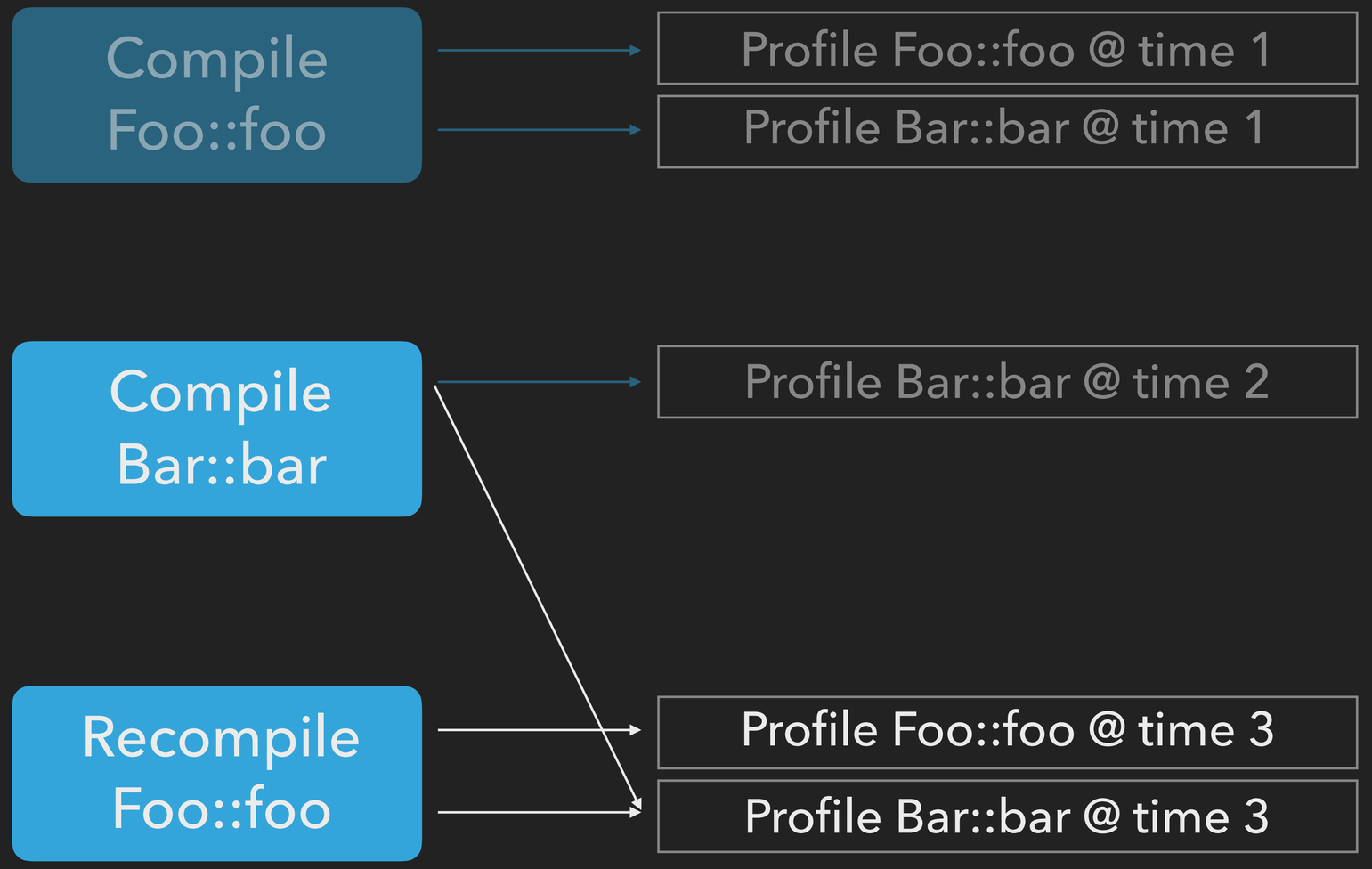
https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/

PICK LAST COMPILE / PROFILE

Reuse last profile for each method

Last profile contains any updates from failed speculations

Produces stable compilations



REPLAYING COMPILATION LOG

WOULD JUST INSTALL TRAPS

TRACK CLASS LOADING AND INITIALIZATION

COMPILATION PREREQUISITES

```
MyClass.getStatic()
```

requires → `MyClass init`

Class 23 SomeClass

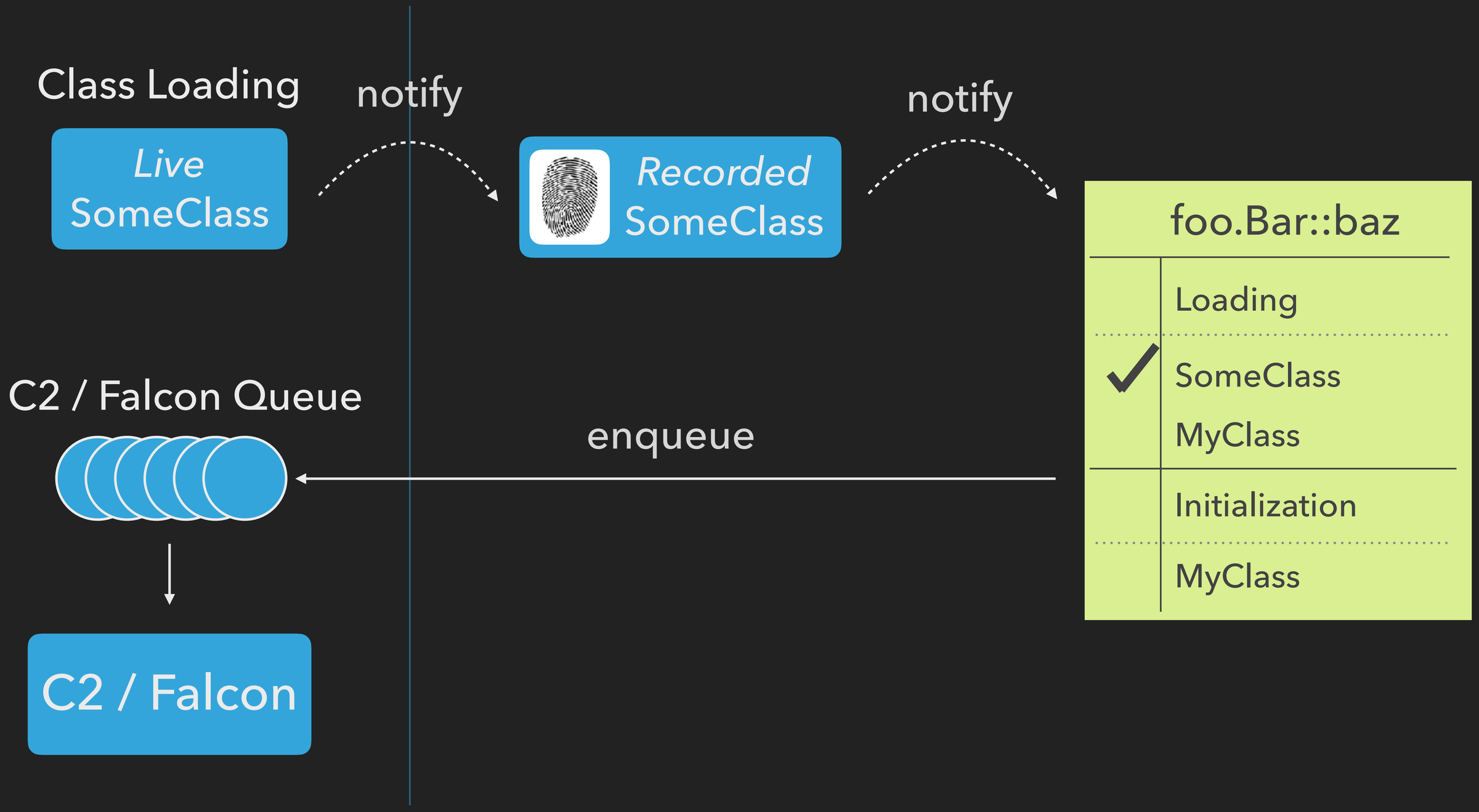
Class 80 MyClass

C2Compilation foo.Bar::baz

{ 23, 80 } loading prerequisites*

{ 23 } initialization prerequisites

* actually linking


Class Loading
Live SomeClass
notify
Recorded SomeClass
notify
foo.Bar::baz
Loading
SomeClass
MyClass
Initialization
MyClass
C2 / Falcon Queue
enqueue
C2 / Falcon

NOT AOT / JIT

ASAP COMPILATION

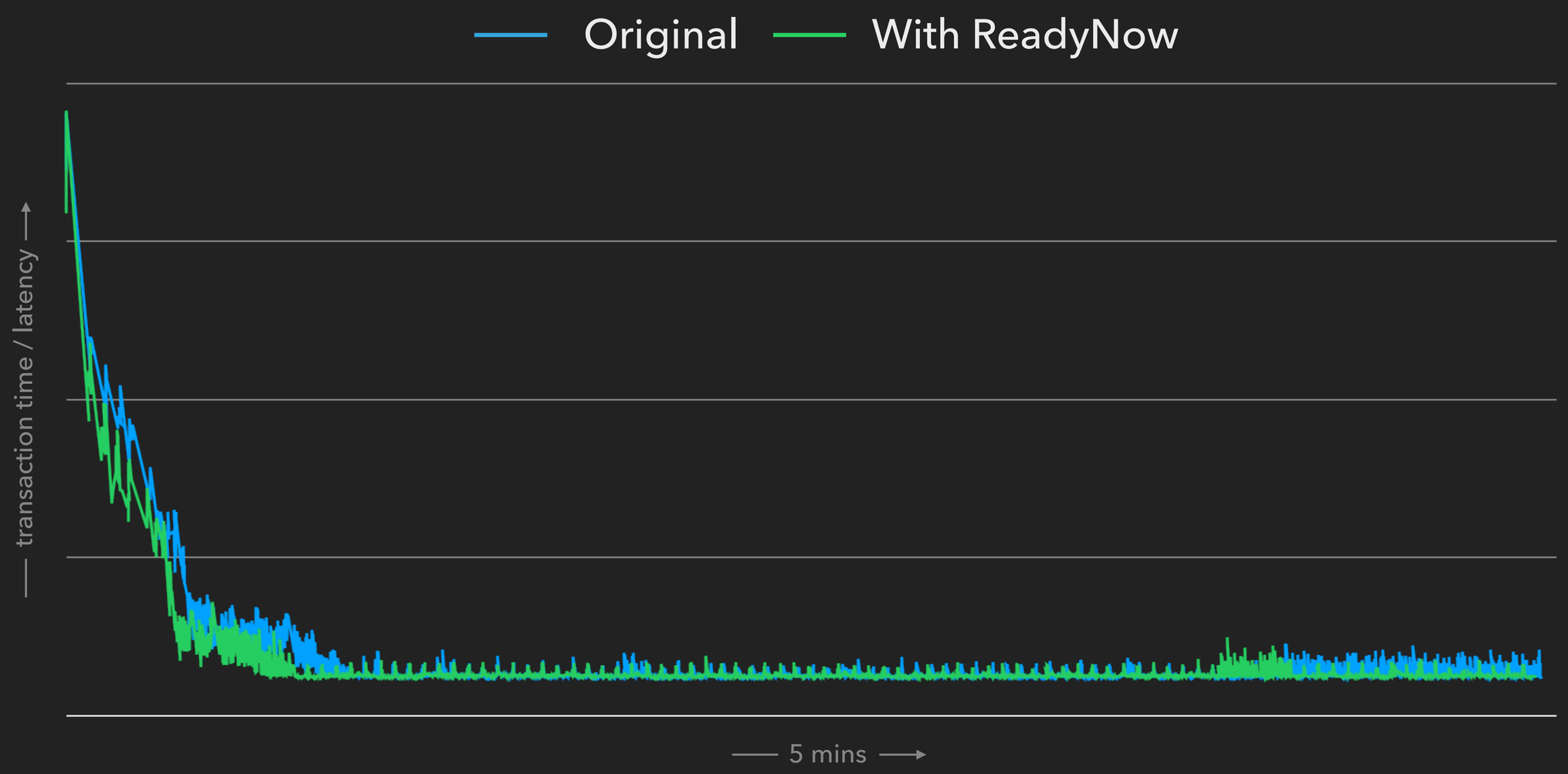
SOUNDS PROBLEMATIC, BUT CONSIDER...

AOT	JIT CACHE	READYNOW
Check Code is Unchanged	Check Code is Unchanged	Check Code is Unchanged
Install Initialization Guards	Wait for Initialization	Wait for Initialization
No speculative optimizations	Speculative optimizations (if profile matches)	Speculative optimizations (if profile matches)

FALSE NOTIFICATIONS ARE OKAY

COMPILER MAINTAINS CORRECTNESS

MAY RESULT IN MORE TRAPS



Peak Perf

-0%

Traps

-60%

STW Deopts

-67%

Not Entrant

-75%

Recompiled

-67%

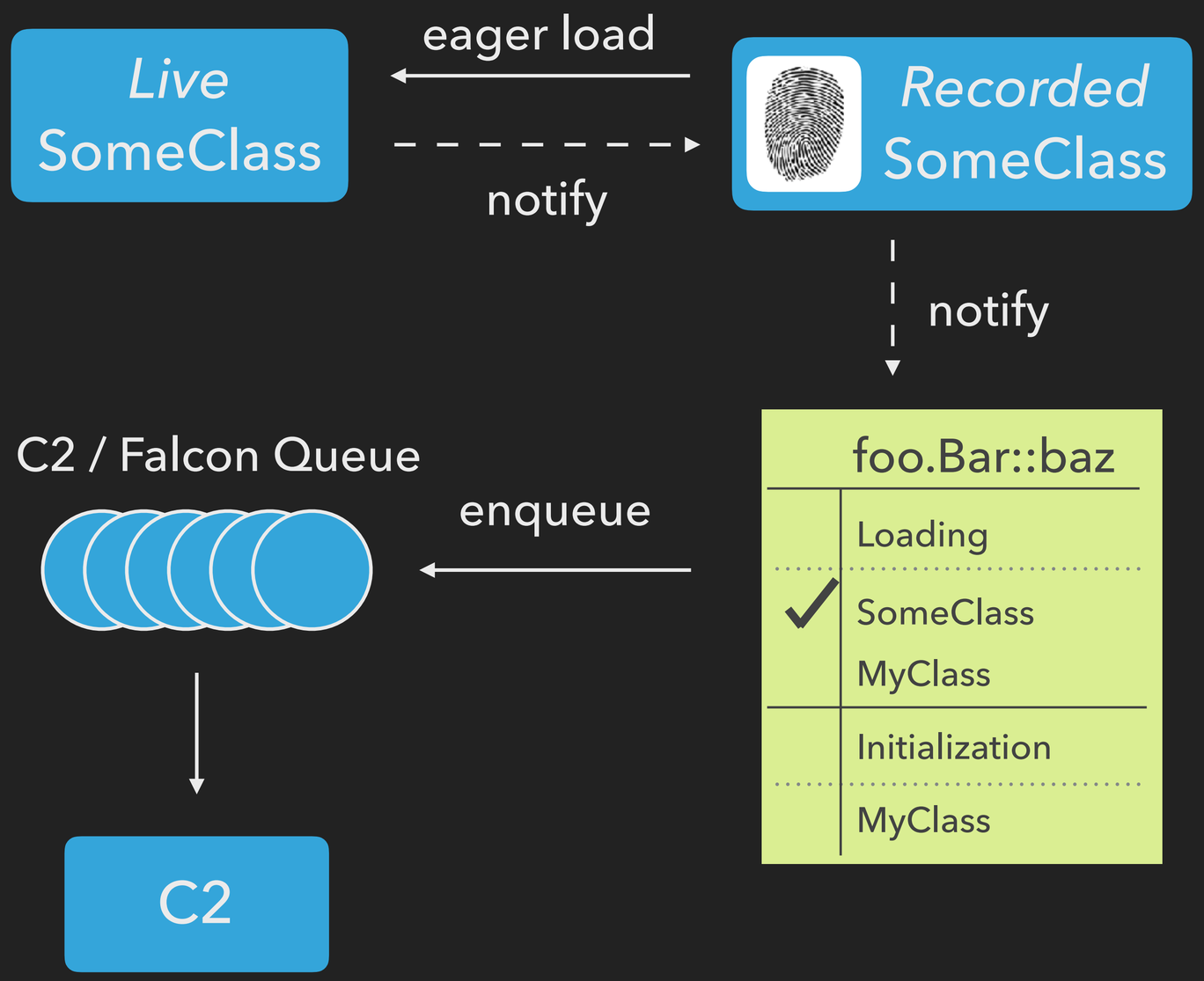
AT START-UP
COMPILATION

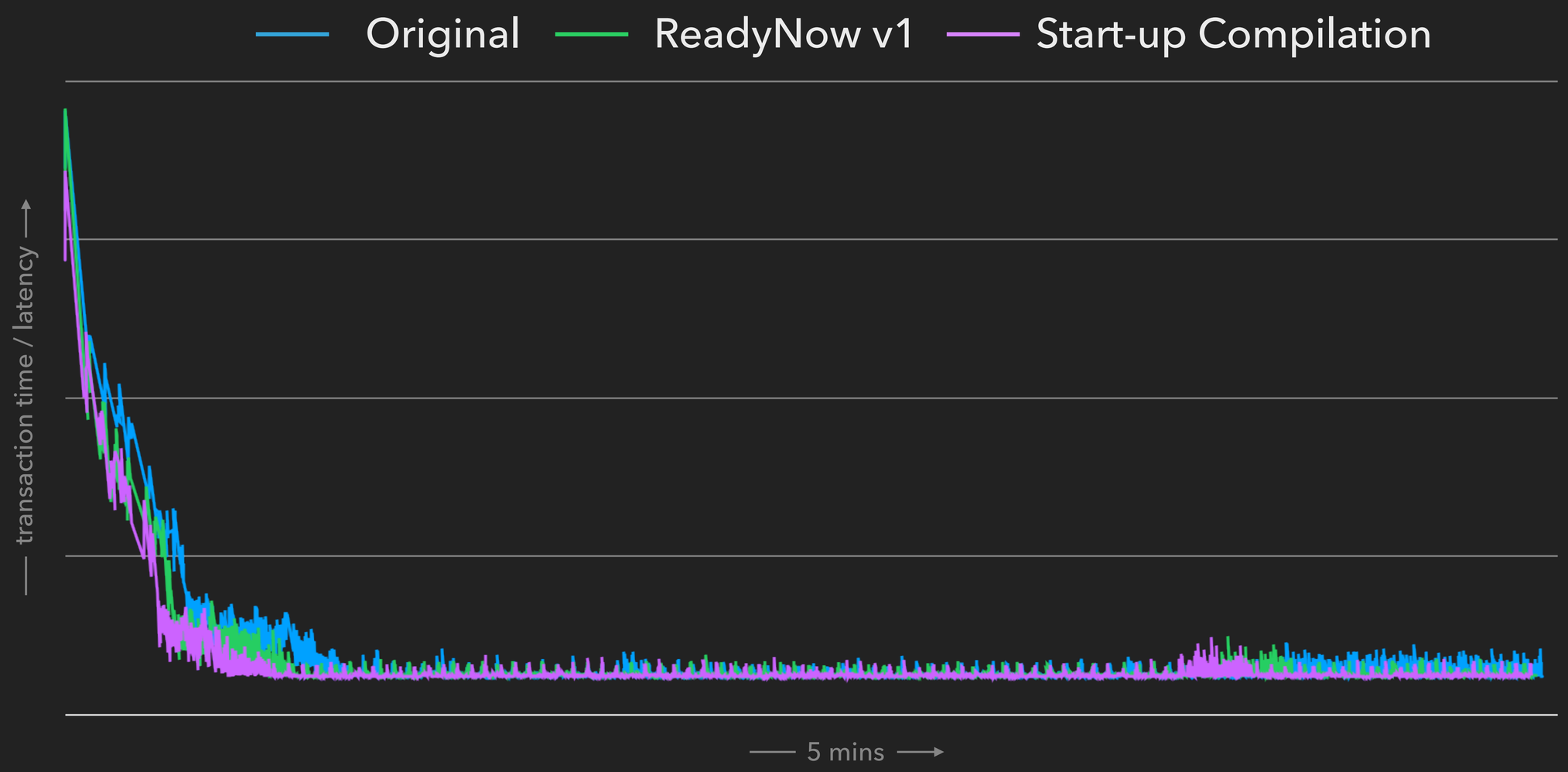
AT START-UP...

Eager Load Classes

Eager Initialize *Most* Classes

Resolve Constant Pool Entries





Peak Perf

+2%

Traps

-49%

STW Deopts

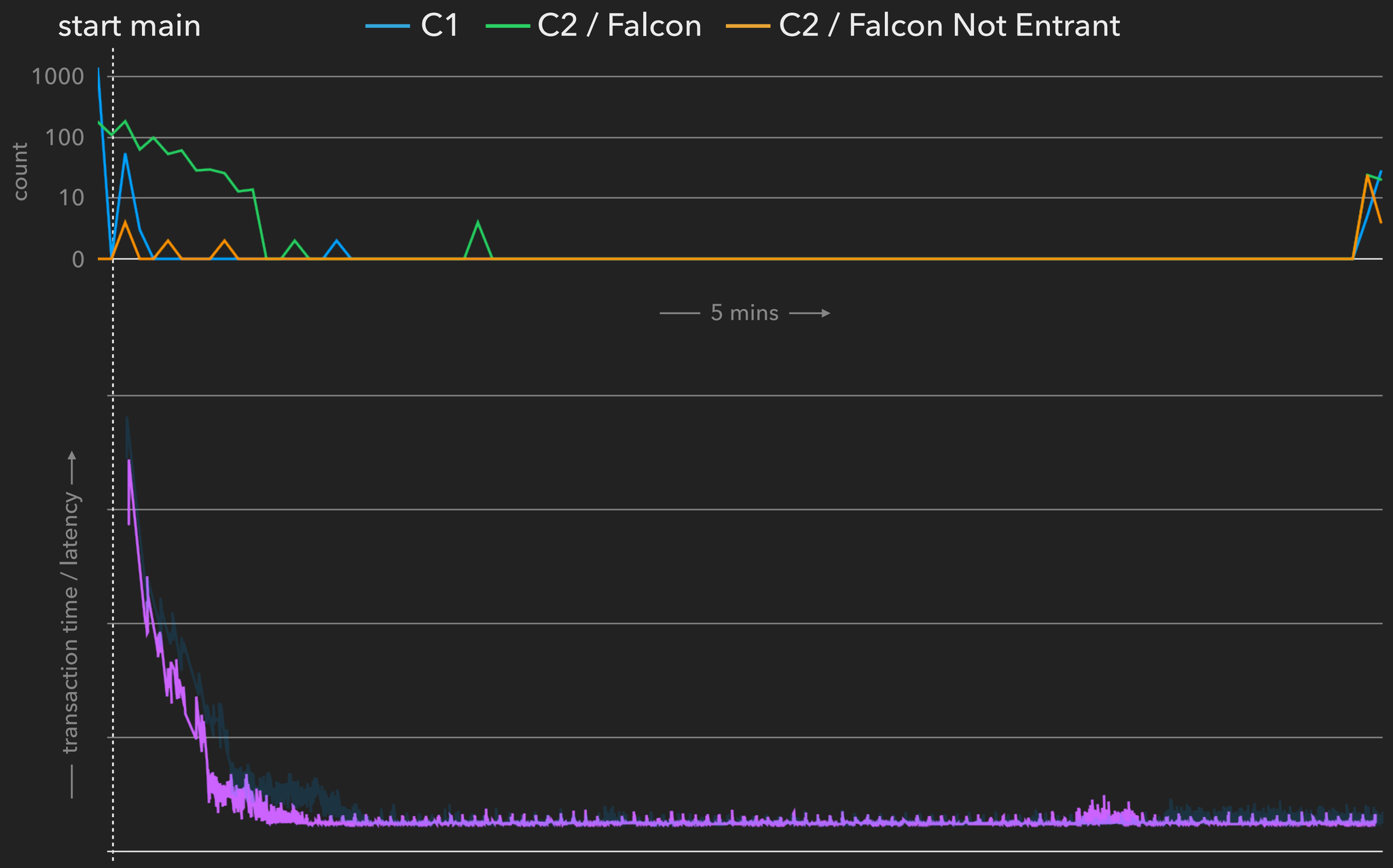
-67%

Not Entrant

-72%

Recompiled

-55%



Peak Perf

+2%

Traps

-49%

STW Deopts

-67%

Not Entrant

-72%

Recompiled

-55%

START REASONABLY FAST!

WARM-UP FAST!

STAY FAST!

PITFALLS!

ClassLoader.definePackage

Native Linking

Asserts

Enums

Initialization Order

ENUMS

```
enum MyEnum {
  FOO,
  BAR,
  BAZ;
}
```

```
final enum MyEnum extends Enum<MyEnum> {
  public static final FOO = new MyEnum("FOO", 0);
  public static final BAR = new MyEnum("BAR", 1);
  public static final BAZ = new MyEnum("BAZ", 2);

  private static final MyEnum[] ENUM$VALUES = {
    FOO, BAR, BAZ };

  private MyEnum(String name, int ordinal) {
    super(name, ordinal);
  }

  public static MyEnum[] values() {
    return ENUM$VALUES;
  }
  …
}
```

ASSERTS

```
class Assert {
  void foo(int x) {
    assert(x > 0);
  }
}
```

```
class Assert {
  static final boolean $assertionsDisabled;
  static {
    $assertionsDisabled =
      Assert.class.desiredAssertionStatus();
  }
  void foo(int x) {
    if ( !$assertionsDisabled ) {
      if ( !(x > 0) ) throw new AssertionError();
    }
    assert(x > 0);
  }
}
```

BIG PITFALL #1

CLASS LOADERS

CLASS LOADERS

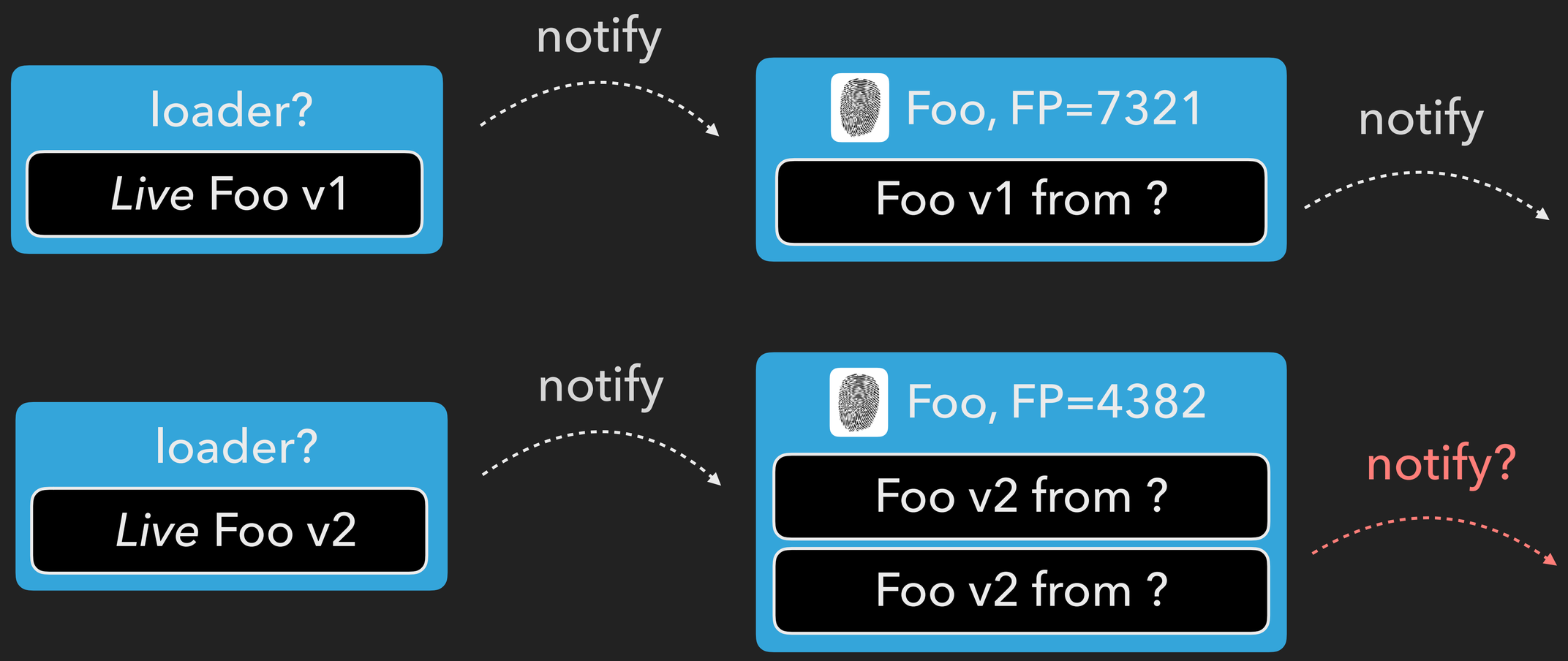
"EVIL" TWINS

```
ClassLoader loader1 =
  new URLClassLoader(new URL[] {
    new File("foo.jar").toURL()
  });
Class<?> fooClass1 =
  loader1.loadClass("Foo");
```

```
ClassLoader loader2 =
  new URLClassLoader(new URL[] {
    new File("foo.jar").toURL()
  });
Class<?> fooClass2 =
  loader2.loadClass("Foo");
```

Indistinguishable After Persistence

CLASS BUCKETS



REMEMBER

FALSE NOTIFICATIONS ARE OKAY

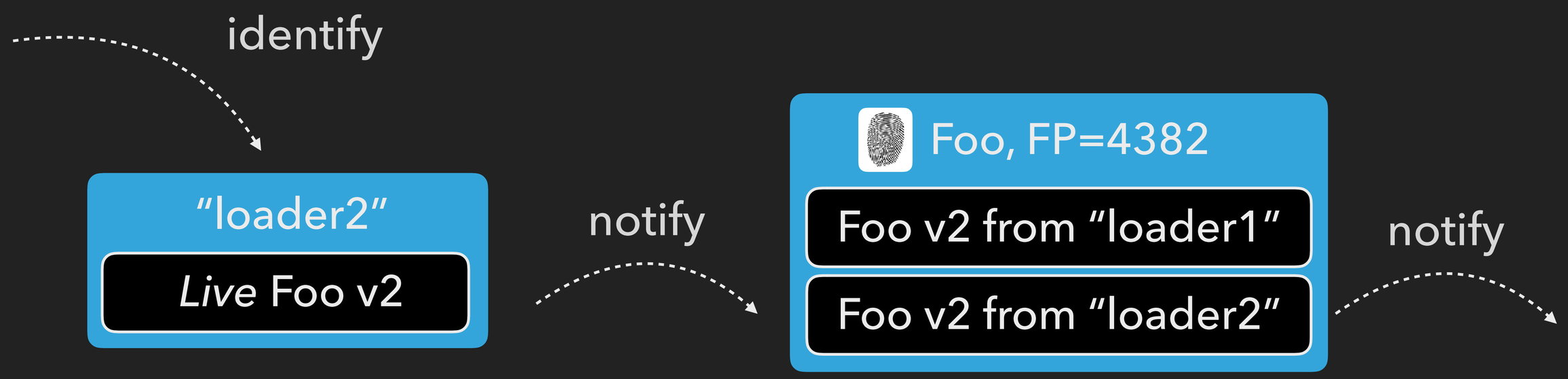
COMPILER MAINTAINS CORRECTNESS

LOADER IDENTIFICATION API

```
ClassLoader loader1 =
  new URLClassLoader(new URL[] {
    new File("foo.jar").toURL()
  });
Zing.identify(loader1, "loader1");
Class<?> fooClass1 =
  loader1.loadClass("Foo");
```

```
ClassLoader loader2 =
  new URLClassLoader(new URL[] {
    new File("foo.jar").toURL()
  });
Zing.identify(loader2, "loader2");
Class<?> fooClass2 =
  loader1.loadClass("Foo");
```

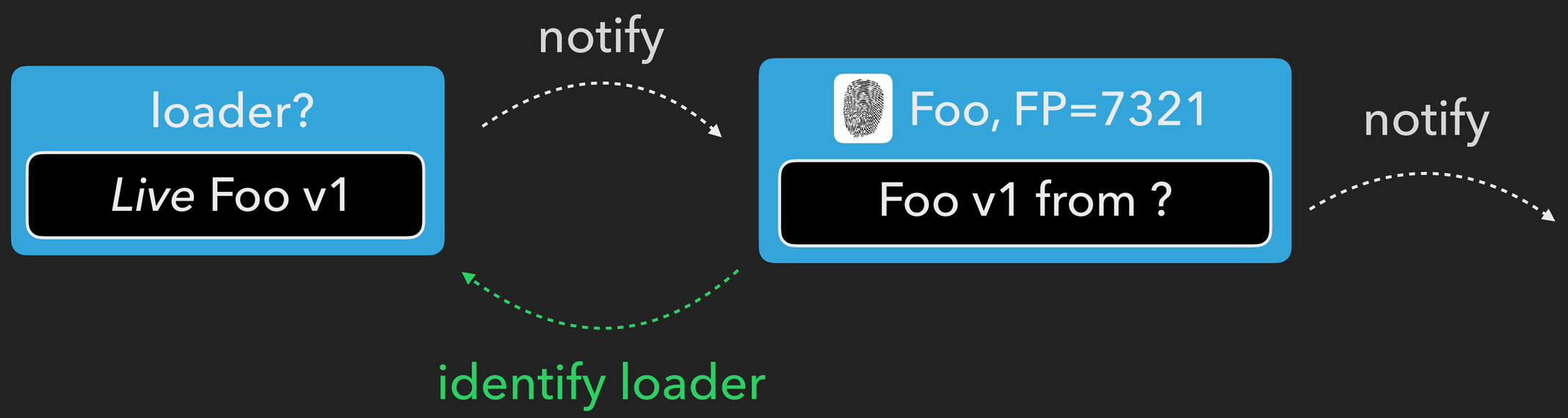
DISAMBIGUATION



"KNOWN" LOADER AUTO-IDENTIFICATION

JDK	COMMON CONTAINERS
Bootstrap ClassLoader	OSGi
Ext ClassLoader	*More to Come*
System ClassLoader	

UNKNOWN LOADER AUTO-IDENTIFICATION



Loader identification triggers
eager load & initialization

BIG PITFALL #2

CLASS GENERATION

EAGER LOADING GENERATED CLASSES?

Generated classes cannot
be loaded immediately

Auto-recognize common code generators -
disable certain problematic behaviors

CLASS GENERATORS: REFLECTION

```
Class<?> klass = obj.getClass();
Method method = klass.getMethod("foo", new Class<?>[] {});

for ( … ) {
  method.invoke(obj);
}
```

java -XX:+TraceClassLoading

```
[Loaded sun.reflect.GeneratedMethodAccessor1 from __JVM_DefineClass__]
[Loaded sun.reflect.GeneratedMethodAccessor2 from __JVM_DefineClass__]
…
[Loaded sun.reflect.GeneratedMethodAccessor999 from __JVM_DefineClass__]
```

CLASS GENERATORS: REFLECTION

GeneratedMethodAccessor1
GeneratedMethodAccessor2
…
GeneratedMethodAccessor537

Big Improvement!

GeneratedMethodAccessor_RN80432DFE
GeneratedMethodAccessor_RN59AD0432
…
GeneratedMethodAccessor_RN432C809E

CLASS GENERATORS: XML TRANSFORM

```
DOMSource source = new DOMSource(…);
TransformerFactory transFactory = TransformerFactory.newInstance();

for ( … ) {
  Transformer transformer = transFactory.newTransformer(source);
  …
}
```

java -XX:+TraceClassLoading

```
[Loaded transformName from __JVM_DefineClass__]
[Loaded transformName from __JVM_DefineClass__]
[Loaded transformName from __JVM_DefineClass__]
```

CLASS GENERATORS: LAMBDAS

```
int prod = Arrays.stream(ints).reduce(1, (r, el) -> r * el);
```

java -XX:+TraceClassLoading

```
[Loaded java.lang.invoke.LambdaForm$DMH/1534030866 from …LambdaForm]
[Loaded java.lang.invoke.LambdaForm$BMH/1927950199 from …LambdaForm]
[Loaded java.lang.invoke.LambdaForm$MH/321001045 from …LambdaForm]
```


https://blog.takipi.com/the-dark-side-of-lambda-expressions-in-java-8/

CLASS MUTATORS: HIBERNATE

java -XX:+TraceClassLoading

```
[Loaded pkg.MyDao$$EnhancerByCGLIB$$2aaa1f58 from ./my-daos.jar]
```

https://dzone.com/articles/cglib-proxies-and-hibernate-lazy-fetching

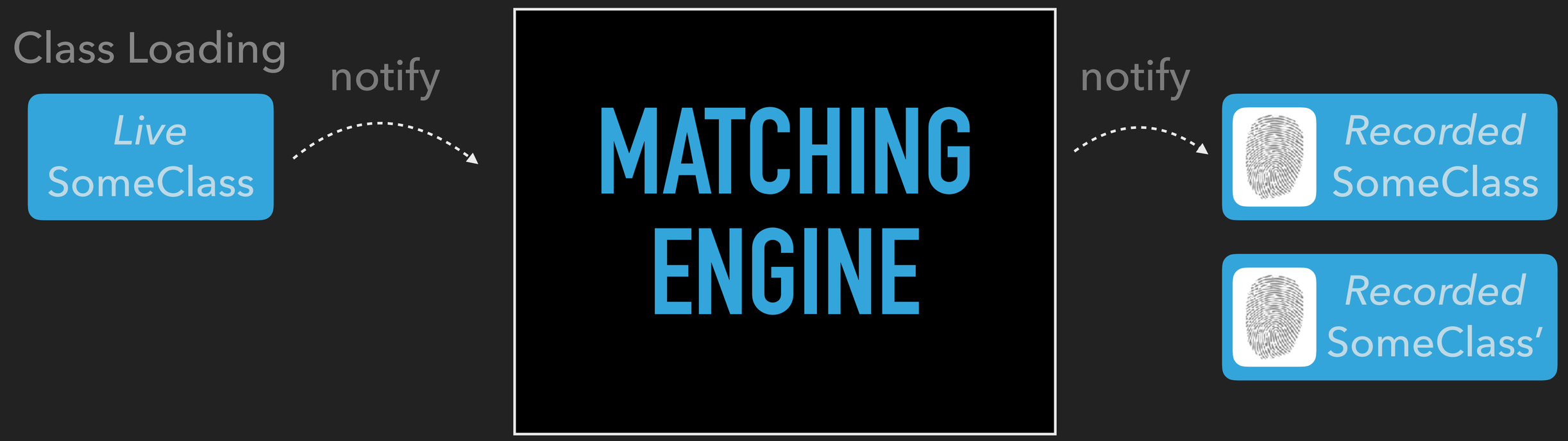
CLASS GENERATION

~~EVAL~~ == EVIL

AOP Rules Engines JSON Libraries Spring

XML Transform ORMs: Hibernate Reflection

Derby Java 8 Lambdas Dynamic Proxies Spark v2



Fingerprint Matching

Structural Matching

Name Normalization

Code Generator Auto-Identification

ClassLoader Auto-Identification

ClassLoader Disambiguation

OTHER MATCHING ADDS RESILIENCY

Method Level Matching

Small Code Changes
JDK / Library Upgrades

Resilient to Inlined Method Change

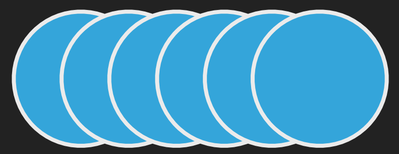
FALCON

Azul's New LLVM-based JIT

DON'T WAIT FOR INITIALIZATION

foo.Bar::baz	
	Loading
✓	SomeClass
	MyClass
	~~Initialization~~
	~~MyClass~~

Falcon Queue

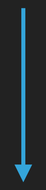


```
MyClass.getStatic()
```

```
if (!vm.is_init(MyClass)) {
  vm.init(MyClass);
}
MyClass.getStatic()
```

INLINE INITIALIZATION CHECKS

```
MyClass.getStatic()
```



```
if ( !vm.is_init(MyClass)) {
  vm.init(MyClass);
}
MyClass.getStatic()
```

Bloats size by 20+%

Slows down by 10+%

Allows for Earlier Stable Compilation

FOLLOW-UP COMPILATION

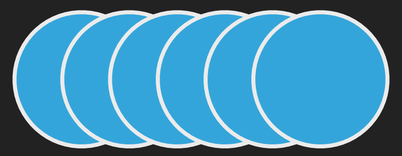
Quick Start Checklist

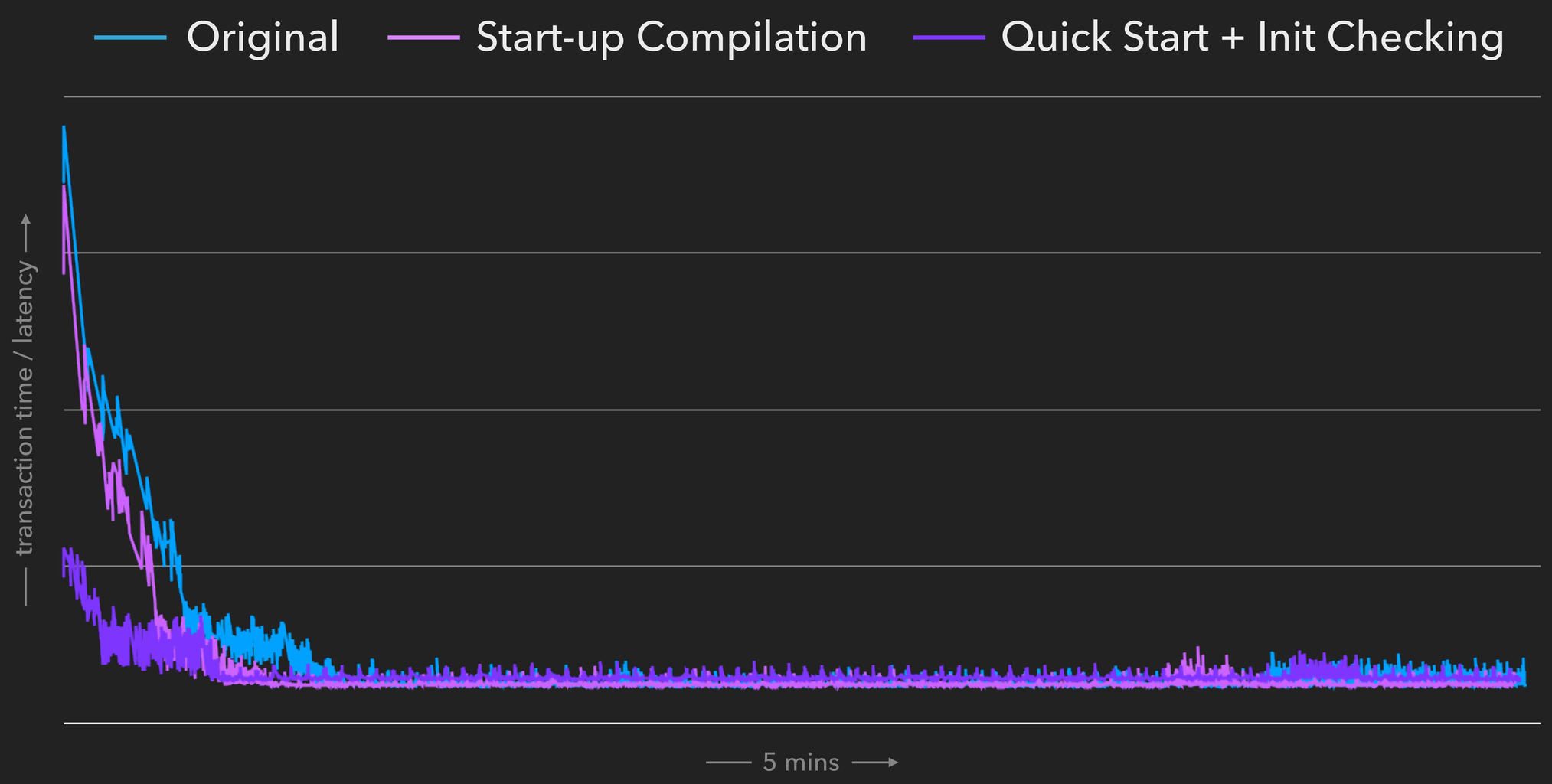
foo.Bar::baz	
	Loading
✓	SomeClass
	MyClass

Maximum Performance Checklist

foo.Bar::baz	
	Loading
✓	SomeClass
	MyClass
	Initialization
	MyClass

Falcon Queue





Peak Perf

-7%

Traps

+40%

STW Deopts

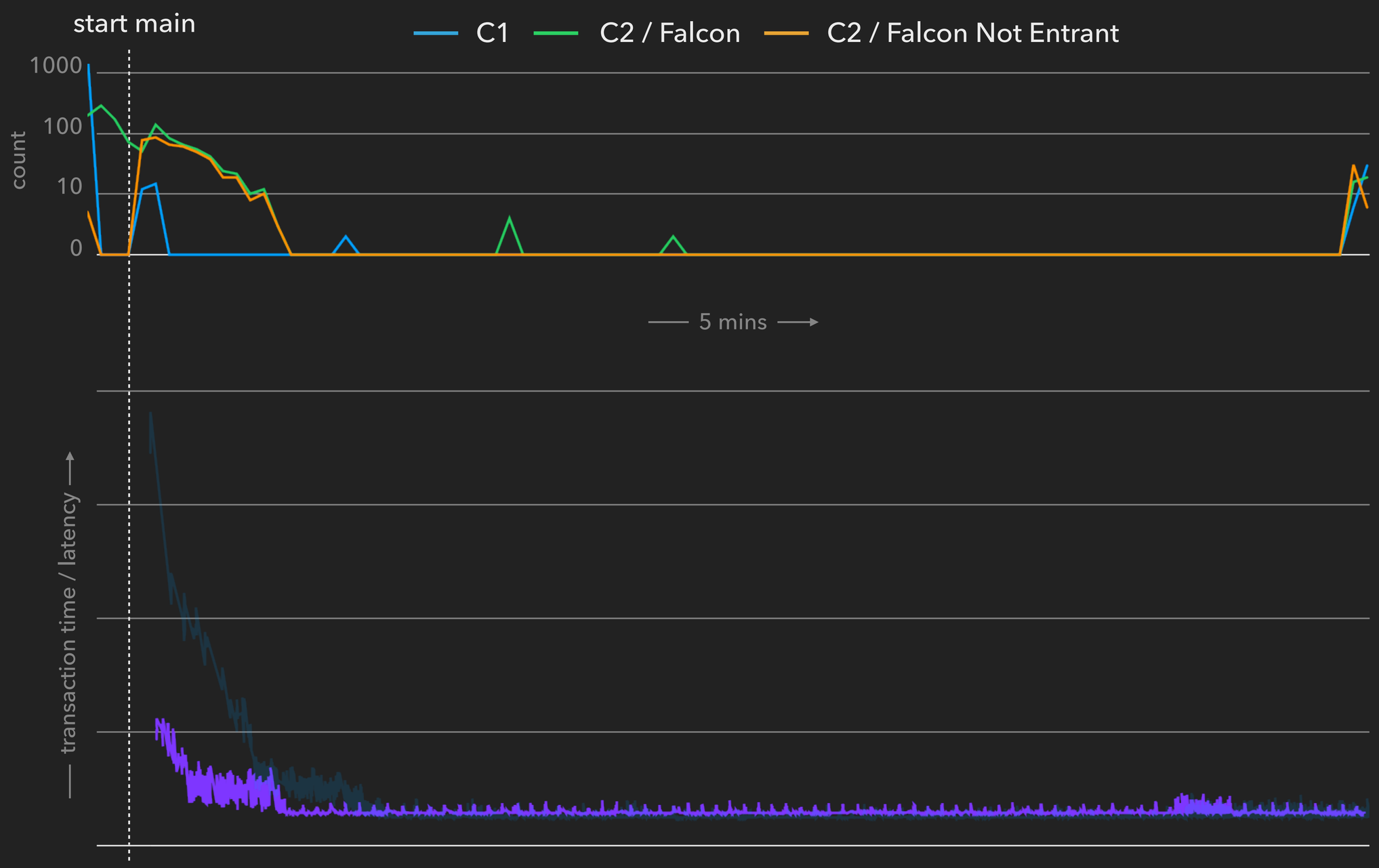
-67%

Not Entrant

+169%

Recompiled

+456%



Peak Perf

-7%

Traps

+40%

STW Deopts

-67%

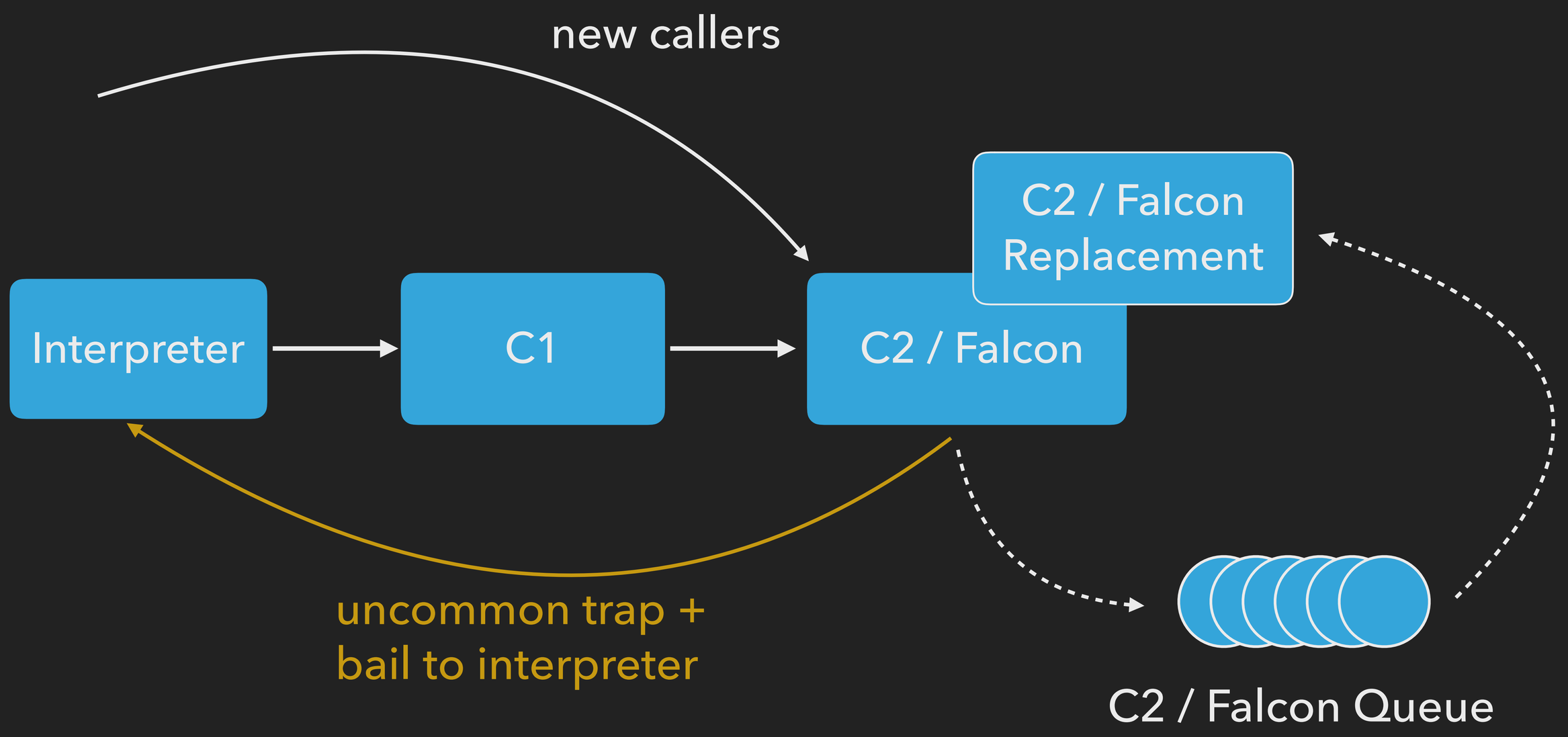
Not Entrant

+169%

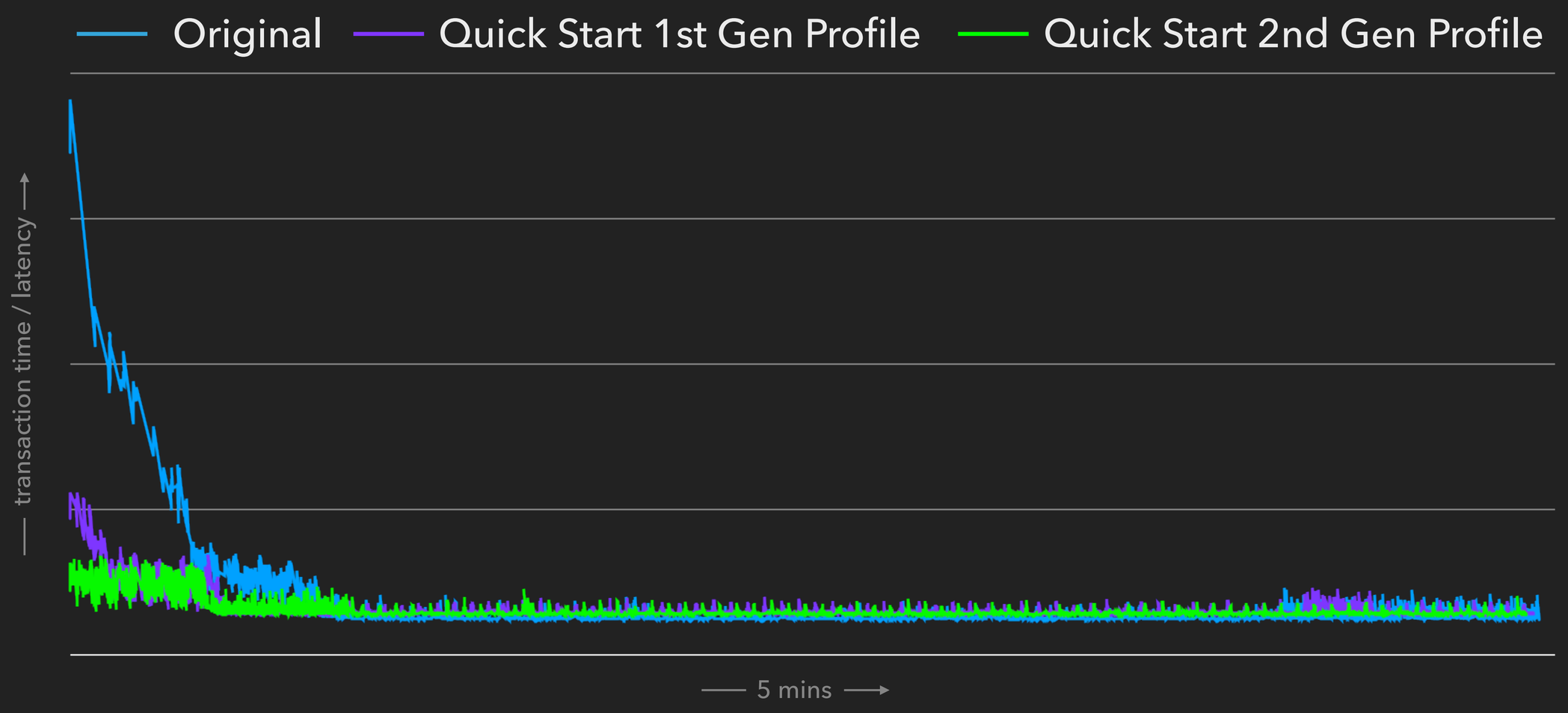
Recompiled

+456%

KEEP ENTRANT + REPLACEMENT



AND WITH ONE MORE ROUND OF PROFILING



RESULTS VARY

60-80% Peak Performance at Start-up

1.5ms - 150ms Initial Transactions

3rd Transaction Usually at Peak

-10 to +10% Impact on Peak Performance

FUTURE BENEFITS OF FALCON

"EASY" TO ADD A JIT CACHE

ReadyNow drives start-up –
hits JIT cache to minimize waiting

ReadyNow's profile cache
backs up JIT cache

ADDING A JIT CACHE

BEST OF BOTH WORLDS

JIT caches misses on
code change, code generators

Profile cache still hits on
unchanged methods

WHAT IS READYNOW?

READYNOW ISN'T...

Conventional

Not an AOT

Not a JIT Cache *Yet*

READYNOW IS...

"Natural" for OpenJDK

Matching Engine

Compilation Log

ASAP Compilation

Profile Cache

REFERENCES

JVM Logging & Inspection Tools

PrintCompilation http://blog.joda.org/2011/08/printcompilation-jvm-flag.html
By Stephen Colebourne

LogCompilation https://wiki.openjdk.java.net/display/HotSpot/LogCompilation+overview
From OpenJDK Wiki

JITWatch https://github.com/AdoptOpenJDK/jitwatch
Created by Chris Newland

REFERENCES

JVM Compilation & Speculative Optimization

Tiered Compilation https://www.slideshare.net/maddocig/tiered
By Igor Veresov

Safepoints in HotSpot JVM http://blog.ragozin.info/2012/10/safepoints-in-hotspot-jvm.html
By Alexey Ragozin

5 Super Useful JIT Optimization Techniques
https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/
By Alex Zhinitsky

Black Magic Method Dispatch https://shipilev.net/blog/2015/black-magic-method-dispatch/
By Aleksey Shipilev

REFERENCES

Other Java AOTs

Java 9 JAOTC
http://openjdk.java.net/jeps/295

Java Goes AOT By Christian Thalinger
http://cr.openjdk.java.net/~twisti/slides/JVMLS%202015%20-%20Java%20Goes%20AOT.pdf

IBM J9 AOT
https://www.ibm.com/support/knowledgecenter/en/SSYKE2_7.1.0/com.ibm.java.aix.71.doc/diag/understanding/aot.html

Excelsior JET
https://www.excelsiorjet.com/tutorials/

THANK YOU

DOUGLAS Q. HAWKINS
LEAD JIT DEVELOPER
AZUL SYSTEMS

@dougqh
dhawkins@azul.com